KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Asking the Right Question: Generating Difficulty-Ranked Questions from Examples

Jinfu Chen

Thesis submitted for the degree of
Master of Science in Engineering:
Computer Science, option Artificial
Intelligence

Thesis supervisor:
Prof. dr. Luc De Raedt

Assessors:
Dr. Damien Sileo
Ir. Thomas Winters

Mentor:
Ir. Thomas Winters

Academic year 2021 – 2022

Preface

I would like to thank my daily supervisor Thomas Winters for his guidance and feedback throughout this year. I would also like to thank my friends that helped me with rating the numerous templates and giving their preference for the difficulty rankings. Finally, I would like to thank my family for their support throughout all the years and especially my mother. Without her I would have missed out on a lot of opportunities in life.

Jinfu Chen

Contents

Abstract

Creating quiz questions is a time-consuming process. Not only is it hard to create a varied set of factual questions, but the question difficulty should also match the knowledge of the participants. This thesis presents an automatic question generation system based on templates. The presented system extracts templates from question-answer pairs with the help of knowledge bases. We fill the extracted templates in with new entities using information about the extracted entities. Results show that the template extraction system performs at its best for question-answer pairs where the questions are short and contain limited entities.

We also sort the generated questions based on difficulty by using a machine learning classifier. The classifier uses popularity and similarity features to output the probability of answering the question correctly. The ranker uses this probability label for sorting the generated questions based on difficulty. Results show that our difficulty ranker requires fewer inserts to obtain the ground truth ranking compared to heuristic difficulty rankers, and that humans prefer the ranking made by our difficulty ranker over rankings made by heuristic difficulty rankers.

These findings show that our described methods could serve as a basis for future quiz generators. A possible extension of our proposed system could be adding a co-creative interface for designing relevant quiz questions with the user.

Samenvatting

Abstract

Creëren van quizvragen is een tijdrovende activiteit. Niet enkel is het moeilijk om een gevarieerde set van vragen te bekomen, maar moet de moeilijkheidsgraad ook passend zijn met de kennis van de deelnemers. Deze thesis stelt een automatische vragengeneratie systeem gebaseerd op gebruik van sjablonen voor. Het systeem extraheert sjablonen uit vraag-antwoord koppels met behulp van kennis basissen. We vullen de geëxtraheerde sjablonen in met nieuwe entiteiten door gebruik te maken van informatie over de geëxtraheerde entiteiten. Experimentele resultaten geven weer dat het sjabloon extractie systeem het best werkt bij vraag-antwoord koppels waarbij de vragen kort zijn en een beperkte aantal entiteiten bevatten.

We sorteren ook de gegeneerde vragen op moeilijkheidsgraad met behulp van machine learning classificatie. Het classificatie syteem maakt gebruik van populariteit en gelijkenis features. De output van het classificatie systeem is de kans dat de vraag juist beantwoord wordt. Experimentele resultaten geven weer dat onze moeilijkheidsgraad rangschikker minder aantal wijzigingen nodig heeft om de ground truth rangschikking te bekomen in vergelijking met heuristische moeilijkheidsgraad rangschikkers, en dat mensen de voorkeur geeft aan onze moeilijkheidsgraad rangschikker over heuristische moeilijkheidsgraad rangschikkers.

Uit deze resultaten zien we dat onze methode gebruikt kan worden als een basissysteem voor toekomstige vragen generatoren dat vragen genereert met een moeilijkheidsgraad. Een mogelijke uitbreiding van ons systeem is de toevoeging een interface voor vragen co-creatie met de gebruiker.

Extractie van sjablonen

Voor de extractie verwacht het sjabloon extractie algoritme een vraag met zijn antwoord als input. De output bevat alle informatie dat nodig is om nieuwe vragen te genereren uit de sjabloon. Het extractie proces bestaat uit twee stappen. In de eerste stap identificeert het algoritme entiteiten om uit de vraag te verwijderen. In de tweede stap legt het algoritme de relatie vast die de sjabloon best beschrijft.

Entiteiten

De eerste stap in het extractie proces is identificatie van entiteiten. Eerst wordt deze voorafgegaan door een voorbewerking. Het algoritme gaat met de NER-tagger van spaCy [11] entiteiten een hoofdletter toekennen als ze toebehoren tot klassen waarvan de entiteiten beginnen met een hoofdletter. Dit zorgt ervoor dat DBpedia Spotlight [17] deze entiteiten niet mist wegens het ontbreken van hoofdletter. Vervolgens gaat het algoritme door het gegeven vraag-antwoord koppel met DBpedia Spotlight en markeert het entiteiten. Het vraag-antwoord koppel moet hierbij minstens twee entiteiten bevatten, waarvan één in de vraag moet voorkomen en het andere in het antwoord. Als laatste bewerking, worden alle gemarkeerde DBpedia entiteiten omgezet tot Wikidata [28] labels, omdat de thesis gebruikt maakt van Wikidata. Het is mogelijk dat een DBpedia entiteit overeenkomt met meerdere Wikidata labels, dus moet er nog duidelijk gemaakt worden welke de juiste label is.

Relatie

Om nieuwe vragen te genereren uit de sjablonen, is het belangrijk om een relatie vast te leggen tussen de geëxtraheerde entiteiten. Deze relatie moet overeenkomen met de semantiek van de sjabloon.

Het algoritme gaat eerst alle directe relaties verzamelen tussen de Wikidata entiteiten in de vraag en de Wikidata entiteiten in het antwoord. Dit legt meteen vast wat de juiste label is indien de entiteit meer dan één label heeft, aangezien dat de kans klein is dat er een directe relatie is tussen twee onjuiste labels. Zodra het algoritme een directe relatie vindt, extraheert het de entiteiten van deze relatie uit het vraag-antwoord. Als er meerdere directe relaties aanwezig zijn tussen de twee geëxtraheerde entiteiten, gaat het algoritme de directe relatie selecteren die best overeenkomt met de semantiek van de bekomen sjabloon. Dit is via vergelijking van woordvectoren tussen kernwoorden van de sjabloon en woordvectoren van de directe relaties. Het resultaat is een sjabloon met twee variabelen.

Sjablonen met drie variabelen

Het is mogelijk dat in de vraag meerdere entiteiten voorkomen, in dit geval gaat het algoritme na of er een directe relatie is tussen de overblijvende entiteiten en de geëxtraheerde entiteiten. Als er een directe relatie aanwezig is, dan extraheert het algoritme deze bijhorende entiteit. Het resultaat is een sjabloon met drie variabelen.

Rangschikking op basis van moeilijkheidsgraad

Een random forest rangschikt de gegenereerde vragen op basis van moeilijkheidsgraad. Om de random forest te trainen, hebben we data verzameld van een quizwebsite genaamd Sporcle [25]. De dataset bestaat uit korte open vragen met een entiteit in de vraag en een entiteit in het antwoord die beide voorkomen in Wikidata. De labels horend bij de dataset zijn percentages van deelnemers die de vraag juist hebben

beantwoord. De random forest gebruikt drie features: gemiddelde globale Google Trends [10] populariteit voor de geëxtraheerde tussen de tijdsperiode 10/03/2017-10/03/2022, het aantal unieke relaties gaande naar de geëxtraheerde entiteiten en het aantal entiteiten met een directe relatie naar beide geëxtraheerde entiteiten. Na training is de random forest in staat om de kans dat de vraag juist wordt beantwoord te voorspellen aan de hand van deze features. De rangschikker gebruikt dit getal om de vragen te sorteren op basis van moeilijkheidsgraad.

Evaluatie

Sjabloon extractie algoritme

Als evaluatie hebben we sjablonen proberen te extraheren uit drie datasets: SimpleQuestions [5], WebQuestions [2] en TriviaQA [12]. Elke sjabloon werd drie keer beoordeeld door verschillende evaluatoren. Zij kregen de instructies om elke sjabloon te markeren als goed (semantisch en syntactisch correct) of slecht. De finale beoordeling van een sjabloon hangt af van welke label in de meerderheid is.

Uit de resultaten zien we dat het extractie algoritme het best werkt bij korte vragen. De twee fouten die het meest voorkomen bij slechte sjablonen is door extractie van een foute relatie en woorden die niet geëxtraheerd werden waardoor de sjabloon maar slechts op één manier ingevuld kan worden. Als verbetering om het aantal slechte sjablonen te reduceren, kan er gekeken worden naar gebruik van andere kennis banken samen met een entity linker dat correspondeert met deze kennisbanken.

Moeilijkheidsgraad rangschikking

Om de rangschikking van makkelijke vragen naar moeilijke vragen te beoordelen, maken we gebruik van het aantal *insert* operaties dat nodig is om van de ene ranking de andere ranking te bekomen. De random forest wordt vergeleken met drie *baselines* in twee scenario's: Hoeveel inserts zijn er nodig om de ground truth orderning te bekomen en welke rangschikking door de vier modellen krijgt de voorkeur.

Uit het eerste experiment is het resultaat dat de random forest model gemiddeld minder *inserts* nodig heeft om de ground truth ordening te bekomen vergeleken met de drie baselines. Uit het tweede experiment zien we dat de random forest model de voorkeur krijgt als de beste rangschikking van makkelijke naar moeilijke vragen.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Terminologies

Abbreviations

QG	Question Generation
MCQ	Multiple-Choice Question
NLP	Natural Language Processing
KB	Knowledge Base
KG	Knowledge Graph
QA	Question Answering or Question-Answer
POS	Part-of-speech
MSE	Mean Squared Error

Terminologies

Key	The correct answer of a multiple-choice question
Distractors	The incorrect answers of a multiple-choice question
To expand a template	To fill in the placeholders of a template
Token	A word
Span	Multiple tokens form a span
Entity	A text span or token that represents a concept
Schema	Description of how to fill the template

Chapter 1

Introduction

1.1 Context

Hosting a trivia quiz is a way to get your friends together and have a fun night. Trivia quizzes mostly consist of different thematic rounds, where all questions within a round is related to a specific theme. These questions can range from different forms: multiple-choice questions, open questions, images, or even sound fragments. Ideally, these trivia questions must satisfy some properties. Questions must be fresh in the sense that participants have not seen these questions before at a different trivia quiz (e.g. *What is the greatest desert on earth?* is a common question that should be avoided). Difficulty should also be appropriate for the participants, to ensure that everyone can at least answer something (e.g. *Who was the* 23^{rd} *US president?* is too difficult for most people). Last of all, questions should have an objective answer in order to avoid discussions (e.g. *Is Japanese anime superior to Western animation?*). As a trivia quiz organizer, creating these kinds of questions is a time-consuming process.

Creating questions is also prevalent for educational assessment. In trivia quizzes questions fit to a specific theme. While for education, questions are used to test whether students understand important concepts in a certain text. For this task, it encounters the same problems as for trivia quizzes. The assessor has to pay attention to a number of things such as: difficulty, relevancy, checking for the right answer etc.

Question generation (QG) is a domain in natural language processing that tries to automate the tedious process of generating suitable questions for activities such as trivia quizzes or educational assessment. Besides generating questions for such activities, QG can also be beneficial for the field of question answering (QA) in natural language processing. QG can be regarded as the complementary task of question answering. In question answering, the task is to generate the right answer given a question and a context. While in automatic question generation, the task is to generate natural sounding questions, given a context or some example questions. It can be seen that one can aid the other. Question answering systems can leverage the questions generated by QG systems for question answering, vice versa QG systems can leverage QA-datasets for question generation.

1.2 Problem Statement

Current QG systems face three problems. One is the lack of user interaction. Having the user interact with the system could be beneficial for generation of desired, high-quality questions. User interaction could allow the user select templates of a certain topic, or even allow users to provide feedback to the generated questions. The second problem is the difficulty of generated questions. Most QG systems generate multiple-choice questions (MCQs) and thus focus on the difficulty of distractors. There are few QG systems that focus on the on difficulty of the question itself. The last problem is how template-based QG systems obtain their templates. In general, there are three main methods for QG: deep learning, syntactic and/or semantic transformation rules and templates. Kurdi et al. [13] and Zhang et al. [31] both use these generation methods in their surveys to categorize QG systems. The first QG method is deep learning, but this comes with a cost of interpretability of such systems. The second QG method is to use syntactic and/or semantic transformation rules. Transformation rules require linguistic knowledge and use syntactic or semantic information. Such systems are hard to understand without proficiency in linguistics, thus again resulting in a lack of interpretability. The third QG method is to use templates. Template QG systems are often paired with knowledge bases (KBs) to obtain factual information to fill in the placeholders of these templates. These templates are often hand-crafted, requiring the user to create templates beforehand, which is tedious and impossible to do when working with a non-domain specific KB, such as Wikidata [28] or DBpedia [14]. Beside these three problems, QG is still a challenging task. The ideal QG system can generate questions of high quality. High quality questions have the properties of being grammatically correct, relevant to the task and difficulty appropriate.

This thesis looks at the following problems:

- Creation of an interpretable template-based QG system
- Creation of questions that are syntactically and semantically correct
- Automating the template creation process
- Estimating the difficulty of generated questions and rank them from easy to hard

1.3 Approach

This thesis takes a template based QG route using Wikidata as knowledge graph (KG). The QG is slightly focused on generating trivia questions, but the system is not constrained on only trivia and can be used to generate general questions as well. For template generation, we exploit existing QA datasets. The template extractor aims to extract useful templates from these existing question-answer (QA) pairs. An entity linker extracts entities to turn the QAs into templates. A direct relation

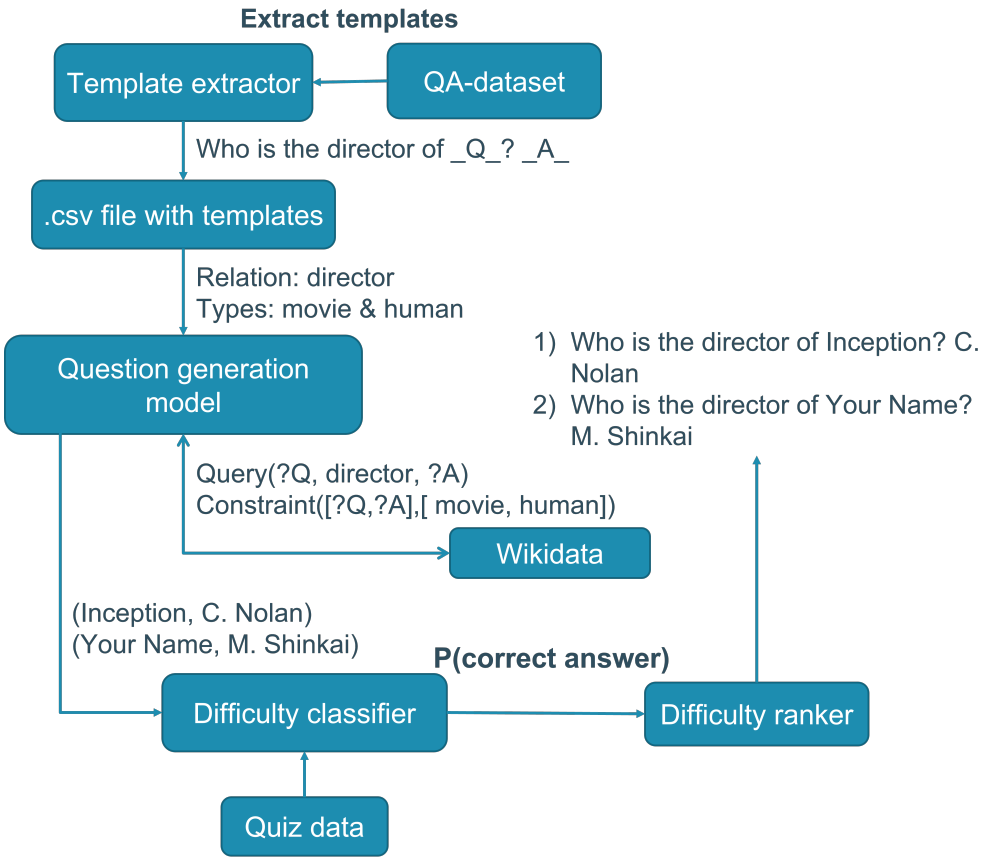

FIGURE 1.1: Approach thesis

connects the extracted entities, and this direct relation is used to fill in the templates. After generating new questions, the difficulty ranker ranks these questions based on difficulty. The ranking uses a pointwise learning to rank approach with random forest as classifier. This classifier uses several features from extracted entities to determine the probability of giving the correct answer to the question. Figure 1.1 schematically visualizes the approach.

1.4 Thesis Structure

In the first two chapters, background and relevant work are thoroughly explained. Chapter 2 explains relevant background to have a basic understanding of the concepts used in this thesis. Chapter 3 looks at existing template-based approaches, where this thesis also draws some inspiration from. The three chapters after related work are about the implementation of the thesis. The bulk of implementation is in Chapter 4, where the approach of extracting templates is explained. These templates are used to generate new questions. Chapter 5 looks at the approach for filling in these templates. Chapter 6 is about ranking these generated questions from easy to hard using a

machine learning classifier. The following two chapters are about the evaluation of the approach taken in the thesis. Chapter 7 discusses evaluation and results of the generated templates, since there are no metrics for measuring template quality, the evaluation is conducted using human evaluation. In Chapter 8, the ranked difficulty questions are evaluated on the accuracy of the ranking. The first part of difficulty evaluation uses metrics, and the second part uses human evaluation. The goal of human evaluation is to check if the difficulty perceived by the system corresponds to the difficulty perceived by humans. The thesis concludes with Chapter 9, in this chapter we summarize the system and look at limitations and future improvements.

Chapter 2

Background

Some background is required to understand some concepts mentioned in this thesis. This chapter provides the necessary information to the knowledge base used in the thesis, natural language processing concepts and random forest for difficulty ranking.

2.1 Wikidata

Wikidata [28] is an open knowledge base consisting of different connected items. Each item is identified using a Q-tag. For example, KU Leuven is identified as `Q833670` on Wikidata. Next to items are properties or relations, properties are identified using P-tags and can be seen as a directed edge connecting two different items. For example, `P495` is the property tag of country of origin. A Q-tag combined with a P-tag can be assigned a value. These are called statements. An example of a statement would be:

`Q833670` (KU Leuven) - `P1075` (rector) - `Q16069692` (Luc Sels)

Another name for statements is subject-predicate-object (SPO) triples. The subject in the example's case is KU Leuven and object is Luc Sels. Figure 2.1 visualizes Wikidata as a graph like structure. This small graph visualizes previous statement along with a new statement connecting Luc Sels and Merksem.

SPARQL is the language for querying resources on Wikidata. The syntax is similar to SQL without the `FROM` clause. An example of a Wikidata SPARQL query that returns all cats with their identifiers can be seen below. Variables in queries are

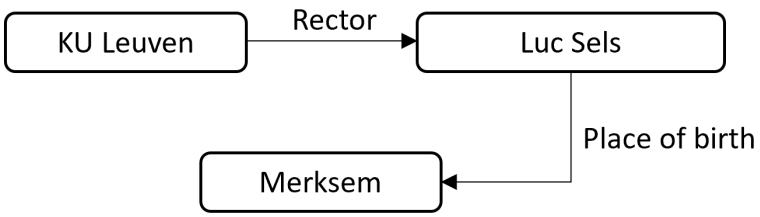

FIGURE 2.1: Wikidata visualized as a directed graph

preceded by a question mark. The `SELECT` clause returns the specified variables as result. Constraints are specified in the `WHERE` clause and acts as a filter. Properties are preceded by the identifier `wdt:` and labels are preceded by the identifier `wd:`

```
# Outputs Q-tag and its corresponding English name
SELECT ?item ?itemLabel
WHERE
{ #item needs to an instance (P31) of cat (Q146)
  ?item wdt:P31 wd:Q146.
  # For English labeling of the Q-tag
  SERVICE wikibase:label
  {bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".}
}
```

Listing 2.1: Wikidata SPARQL cat example [28]

2.2 Random Forest

The difficulty ranker (Chapter 6) uses random forest as a classifier. The basis of random forests are decision trees. This section explains decision trees as a classifier and how it leads up to random forests. Knowledge of feature vectors is required for this section. Feature vectors are vectors consisting of values of different features. The classifier uses these features to build a classification model and to predict the output label.

2.2.1 Decision Trees

Decision trees are trees where nodes are annotated with a feature and the edges are annotated with possible values of that feature. To apply this classification model to a given feature vector: Start at the root node, at each node take the path that corresponds to the value of the example's feature. Leaves represent the prediction made by the decision tree. Figure 2.2 shows a decision tree for feature vectors consisting of three Boolean features: `Leftover work`, `Weekend` and `Deadline soon`. The decision tree's prediction is based on these three features, and outputs if the student should or should not study.

The split criterion determines the feature to split on, information gain is a possible split criterion. When using information gain, the tree is built top-down by selecting the feature that introduces the most information gain. Another possible split criterion is Gini impurity.

2.2.2 Bootstrap Aggregating (Bagging)

Bagging (Figure 2.3) is an ensemble method in machine learning where the ensemble's output is based on the output of several different classifiers. For training these different classifiers, data is randomly sampled from the whole training dataset with replacement till the amount of data sampled is equal to the amount of data in

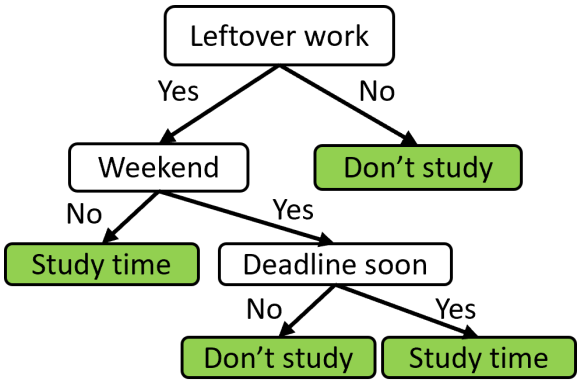

FIGURE 2.2: Decision tree example

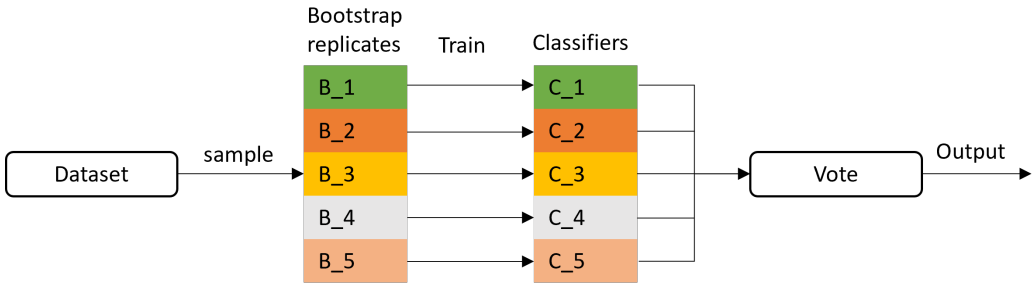

FIGURE 2.3: Bagging illustrated

the whole training dataset. This sampled dataset is called the bootstrap replicate. Multiple bootstrap replicates are created, and each classifier is trained on a different bootstrap replicate. At prediction time, each of these classifiers make their prediction and the output is based on majority vote.

Random forest is a bagging method where the classifiers are decision trees, but at training time, at each split, only a subset of features are considered as the split feature. This is to ensure that the decision trees are not necessarily built in the same way. Figures 2.4 and 2.5 show the difference between a classic decision tree and a decision tree that is used in a random forest. In the decision tree, all features that are not yet part of a split can be selected as the feature to split on. For the random forest decision tree in this example, we randomly withhold one feature at each split.

2.3 Pointwise Ranking

Pointwise ranking is an approach of ranking within learning to rank where items are ranked using the output label of a classifier. Pointwise ranking is a two-stage process. In the first stage, a classifier trains on data that outputs the desired label that is used to rank on. In the second stage, given a list of items to rank, let the trained classifier predict that label for each of the items and rank them based on it.

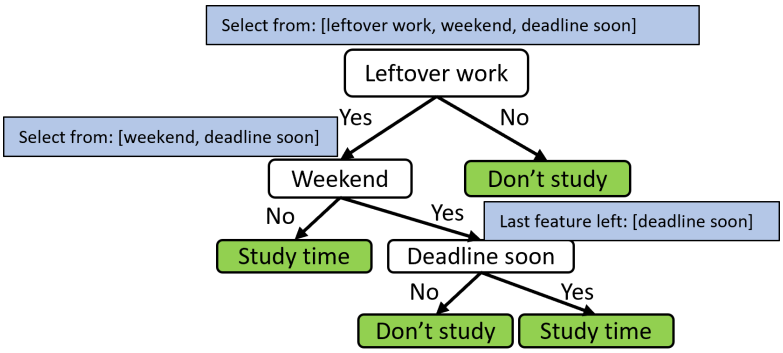

Figure 2.4: Features visualized of decision tree split

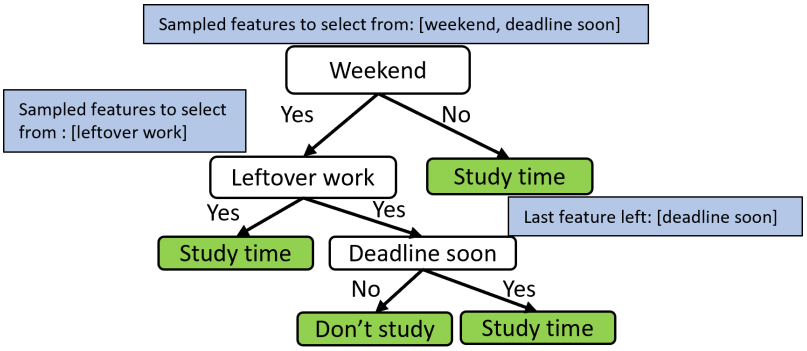

Figure 2.5: Features visualized of a decision tree split in a random forest

For example, the task is to rank a list of food in ascending tastiness. The training set consists of food features and a tastiness factor. A classifier is trained to predict this tastiness factor. Given the list to rank:

`[apple,durian,mango,dragon fruit]`

the classifier first predicts the tastiness factor for each food item:

`[0.5,0.2,0.7,0.6]`

The list gets sorted based on the prediction label and thus the ranking is:

`[durian,apple,dragon fruit,mango]` .

2.4 Natural Language Processing Concepts

2.4.1 Named Entity Recognition (NER)

Named entity recognition (NER) is a task in natural language processing where entities are marked with a class tag. Entities refer to a token or text span that represents a concept. A NER tagger assigns NER tags, the classic NER tags are: `PER` (person), `ORG` (organization), `LOC` (location) and `GPE` (geo-political). Tagging can be seen as a sequence labeling task, where each word of an input sentence receives a

Tokens	My	name	is	John	Smith
BIO tags	`O`	`O`	`O`	`B-PER`	`I-PER`

TABLE 2.1: Illustration of BIO-tagging of sentence *My name is John Smith.*

Tokens	I	like	computer	games
POS tags	`PRON`	`VERB`	`NOUN`	`NOUN`

TABLE 2.2: Illustration of POS tagging of sentence *I like computer games.*

label. Because tags can span over several tokens, and sequence labeling tags token per token, sequence labeling makes use of BIO-tagging. In BIO-tagging, every word receives a label `B` (begin), `I` (inside) or `O` (outside). An example can be seen in table 2.1 for the sentence *My name is John Smith.* Other approach to NER tagging can be parse tree labeling.

Entity tags are helpful in natural language tasks. In question answering, tagged entities are more likely to be the answer to the question. These tags can also be used to pre-identify where possible entities are for entity linking.

2.4.2 Entity Linking

Similar to NER tagging, entity linking also tags entities, with the addition that tagged entities are associated with an item within a knowledge source. This thesis uses DBpedia Spotlight [17] as an entity linker. DBpedia Spotlight works in four-stages. In the first stage, DBpedia Spotlight finds candidate items present in DBpedia. During the second stage, these candidate items are mapped to DBpedia items. In the third phase, each entity gets disambiguated using the context around this entity. The last phase is where the user can tune some parameters for filtering. Using DBpedia Spotlight as an entity linker is limited to DBpedia items, but spaCy [11] also offers a trainable entity linker to train it to a knowledge base of choice.

2.4.3 Part-of-speech (POS) Tagging

Part-of-speech tagging resolves syntactic labels of tokens. POS tags show how tokens are used within a sentence. Methods for POS tagging can be token-based tagging, where each token gets a POS tag assigned, or sequence tagging, where a whole sequence gets assigned a sequence of POS tags. Table 2.2 shows an example of token-based tagging.

Chapter 3

Related Work

This chapter highlights related work that is relevant to this thesis. Since this thesis focuses on using templates and knowledge graphs for question generation, neural, syntactic, and semantic works are not discussed, because the techniques are completely different and makes it hard to compare to the system described in the thesis. The chapter starts with the works of template-based QGs and finishes with a summary of the works and their properties.

3.1 Template-based QG

Template-based QGs make use of placeholders that can be filled in to generate new questions. KBs are the primary source for obtaining information to fill in these placeholders. KBs can range from general KBs like Wikidata to domain specific KBs.

3.1.1 MCQ

Sherlock

Liu and Lin [16] proposed a QG system for educational purposes and a way to control the difficulty of generated questions. The system works with templates that are defined beforehand and are made into MCQs. In the paper, these templates are filled in with data from DBpedia and British Broadcasting Corporation. The difficulty estimation is between the distractors and the key, using linked data semantic distance (LDSD) [19]. Distractors are clustered into three difficulty levels (easy, medium and difficult) using K-means clustering based on their LDSD with the key. Results show that LDSD is a suitable metric for measuring difficulty. This way, Sherlock can for example generate a question asking a question containing images (Figure 3.1).

In the following year, Lin et al. [15] improved on Sherlock by changing the difficulty estimator. Instead of using LDSD, they proposed another algorithm using a hybrid semantic measure called TF-IDF (LD) [15] to categorize the questions. Using this new algorithm, the clustering accuracy improved drastically compared to other baselines.

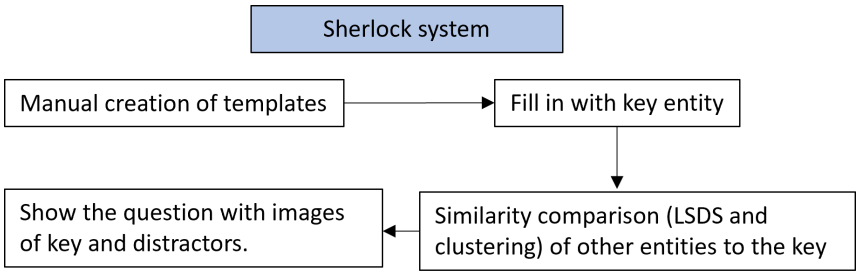

Figure 3.1: Sherlock [16] system visualized

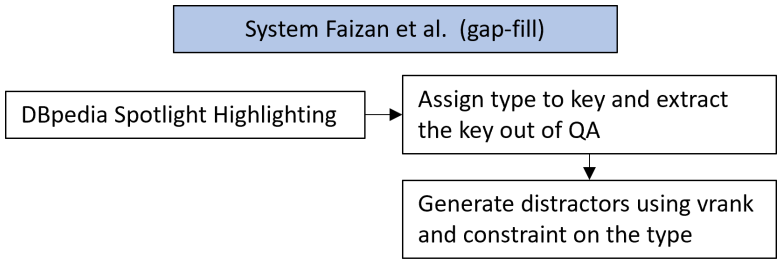

Figure 3.2: Faizan et al. [8] system visualized for gap-fill questions

Multiple Choice Question Generation for Slides

Faizan et al. [8] created a system (Figure 3.2) that generates three types of MCQs by making use of DBpedia Spotlight to identify DBpedia resources. The first type of questions are gap-fill questions, the second type are generic MCQs asking for the type and the last variety consists of Jeopardy-style questions. The system executes a different algorithm depending on which type of questions the user desire. Gap-fill questions are created by replacing the identified DBpedia resource with a blank. For generic MCQs, the system queries the type of the resource, and sub-classes of this type are selected as distractors. For Jeopardy style questions, the system also queries two SPO triples of the identified DBpedia resource along with its type. These Jeopardy style questions are verbalized using handcrafted templates.

The concept of type depth is used to generate distractors of appropriate difficulty. Keys get a shallow type assigned (e.g. person) for easy and deep type for hard (e.g. politician). The idea is that deep types are less likely to be known compared to shallow types. The more specific the type of the key, the more difficult the distractors are. Distractors are either entities of the same type as the key for resources, or sibling types in case the key is a type itself. In case of Jeopardy style questions, the difficulty depends on the chosen two SPO triples. The well-knowness of a SPO triple is measured using vrank value [26], and should correlate with probability of knowing the right answer.

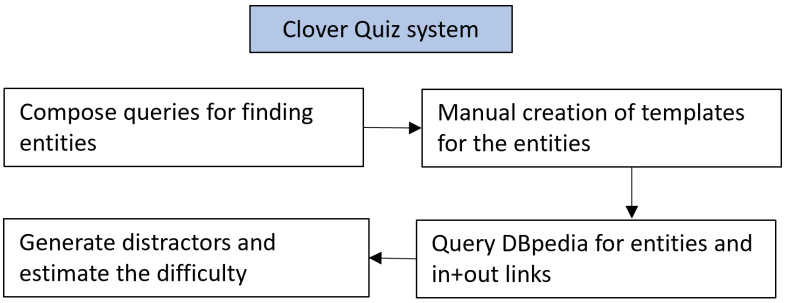

FIGURE 3.3: Clover Quiz [27] system visualized

Clover Quiz

Gorgojo [27] made a trivia game based on information found in DBpedia. The QG is a two stage QG. In the first stage, a human selects interesting entities and concepts from DBpedia and queries them. Additionally, similar entities to the queried entity are retrieved as well. The second stage of question generation is to fill in the templates, these templates are handcrafted beforehand. Question difficulty is connected to its popularity, and this is measured using incoming and outgoing links of the DBpedia entity. The generated questions are MQCs, where the distractors are entities of the same class. Figure 3.3 summarizes the most important properties of Clover Quiz.

The generated questions also have a difficulty associated with it depending on three properties. The first of which being popularity score `popularity` = `outgoing_links` + 10∗`incoming_links`, the more popular an entity is, the easier the question should be. The second is a subjective difficulty score given to templates. The last one is distractor difficulty, distractors that are more similar to the key are considered harder questions.

Gorgojo [27] concluded several findings from the application, but only two are relevant to this thesis: DBpedia can be easily used for new applications and the popularity score is a suitable difficulty heuristic.

Thematic Question Generation over KBs

Raynaud et al. [21] created a template-based QG system utilizing KBs for generating MCQs. The approach leverages Wikipedia structure for topic selection. Using lists, categories, portals and outlines of Wikipedia, a range of topics can be obtained. Each topic contains Wikipedia resources, and a filter for relevant resources is constructed using a recursive crawl. The system also expands that initial set of resources with other similar resources by using latent semantic analysis (LSA). Raynaud et al. [21] proposed two possibilities to obtain templates for QG. Either the user constructs the templates manually or uses the QA templates generated by Abujabal et al. [1]. A 3-tuple (Figure 3.4) defines templates in this paper. The first element, called stems, represents QAs where the templates are generated from. The second element contains

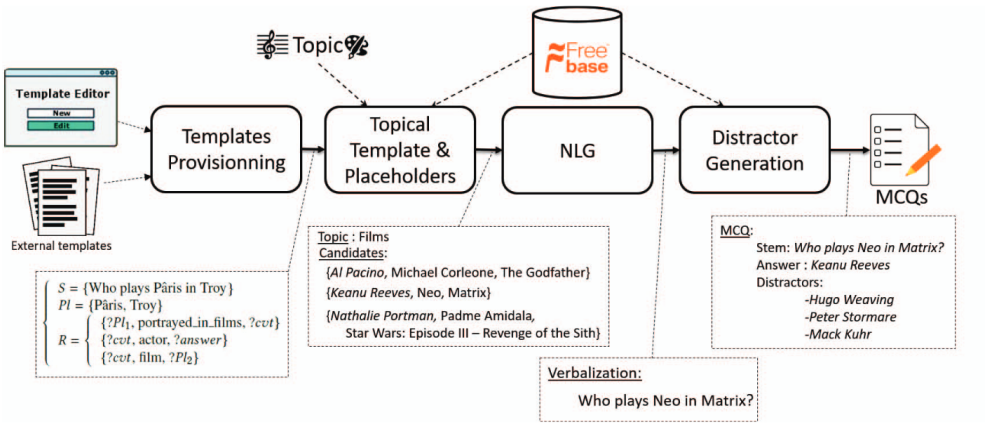

Figure 3.4: System of Raynaud et al. [21] , image taken from *Thematic Question Generation over KBs* [21]

placeholder entities, from which should be extracted from stems. The last element is a sub-graph of KB to find new entities to fill into the placeholders. Templates are then assigned a specific topic by checking the types of entities that can be filled into the placeholders. In case that the types are identical, the template is assigned a topic category, otherwise the topic is calculated at QG time depending on the entities in the question. With the topic known for each template, a formula is designed to select the suitable template and placeholder entities given a topic.

Distractors for MQCs must be of appropriate type and relevant to the question. Type appropriateness is achieved by selecting entities that also match the template. In case this is not possible, distractors are selected based on the type of the key. Irrelevant distractors are filtered out using PageRank and LSA. All these elements combined form the system in Figure 3.4.

Results showed that on average 69% of the generated question is related to the assigned topic. Question quality scored 73%.

3.1.2 Open Questions

Question Generation from a Knowledge Base

Chaudhri et al. [7] developed a QG system for a domain specific KB consisting of biology-themed concepts. The goal is to let the user input some question and the system should suggest questions that are similar to the user inputted question such that the returned questions are questions that the KB can answer. Besides letting the user input questions, the system is also able to generate questions based on the entity's page.

Question generation is tackled using manual construction of templates. For their domain specific KB, Chaudhri et al. [7] concluded that questions that could be answered by the KB can be generalized to 30+ templates. With these templates,

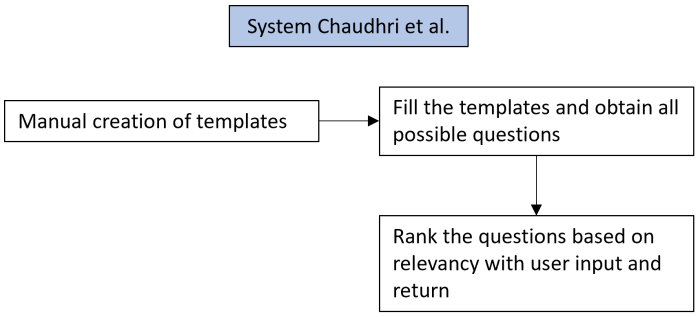

FIGURE 3.5: System of Chaudhri et al. [7] visualized

the KB is crawled to generate instances of the templates. For each type of template, there exists different rules to fill in the template. For example in their work [7], for the template `What is the R of X?` if `X` is the root node in the knowledge graph, the relation connecting the nodes `X` and `Y` is filled into placeholder `R`. Verbalization is performed depending on the number of values that R has. For example in their work [7], the relation *has-part* for the root node `Mitochondria` has two values: `Mitochondrial-DNA` and `Mitochondrial-Membrane`. The question becomes: *What are the parts of the mitochondria?* These questions are generated beforehand and result in a database consisting of more than 20k questions.

For outputting relevant questions, the system of Chaudhri et al. [7] has to rank the generated questions according to relevancy to the user's query. First, the input is scanned for entities and relations to find relevant questions. The system then retrieves questions that relate to the identified entities and relations. There are three scores for ranking these retrieved questions. One is based on the question type. The second one is the rank within a (KB) concept. This is the score is based on the length of the questions. The last is the rank within the questions for a concept for that question type, also based on the length of question. The final score is the product of these three rankings.

Results showed that the system could generate 70% of all potential questions generated by two biology teachers over six paragraphs. Two biology teachers also rated the quality of 376 unique generated questions over a selected page, 28.7% were useful questions, 43.6% were mediocre questions and the rest were ranked as questions of poor quality. Figure 3.5 shows the system by Chaudhri et al [7].

QG from a KB with Web Exploration

Song and Zhao [24] created a system that uses Google's search engine for generating additional questions (Figure 3.6). Templates are created manually for each predicate in the KB. With the templates ready, the system generates an initial set of questions using the entities from the KB. After obtaining this initial set of questions, these questions are used as queries in Google. When searching from something, Google shows a list of questions that other users also asked that are related to the search

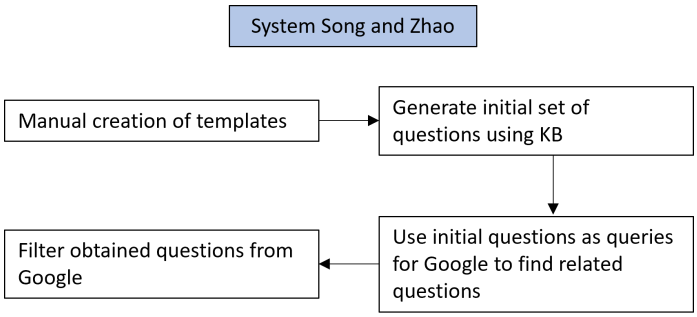

Figure 3.6: System of Song and Zhao [24] visualized

query. These related questions can be used to obtain new questions without updating the KB or crafting new templates.

The questions obtained from Google's search engine may not be relevant or fluent. The first filter checks for domain relevance. Domain relevance is defined as $cos(v(q), v(D_{in}))$ with $v(.)$ as the document embedding and D_{in} as the initial question set. The second filter checks for fluency with the average language model score $(skip\text{-}gram(q)\ /wordcount(q))$. The system applies a threshold for both scores to filter out questions.

For evaluation, Song and Zhao sampled 500 random statements from Freebase [4] to generate questions. After obtaining additional questions from Google, the top 500 questions were selected using the average language model as index to compare to the system of Serbal et al. [22]. Evaluators rated the questions on a scale of 4 on grammar and naturalness. Results showed a 0.17 score increase on grammatical and naturalness compared to the work of Serban et al. [22], which was the state-of-the-art KB QG at the time.

Knowledge Questions from Knowledge Graphs

The work by Seyler et al. [23] shares some similar goals to this thesis. Their goal is to allow the user to determine the topic of the question and its difficulty (easy or hard). New questions are generated by using information of Yago2s [3] KG.

The first step is to generate a query based on the user's given topic. The query used in Seyler et al. [23] is similar to a Wikidata statement. It consists of two entities connected by a relation. The system selects a Yago2s entity that corresponds to the topic as answer entity. These queries have the answer entity as subject or object and are replaced by a variable. Queries are used to generate questions later on. Before the question can be generated, the type of the answer entity must be selected. The questions use the type of the answer entity to refer to. For example, when asking: *Who the rector of KU Leuven in the year 2022?*, rector refers to Luc Sels. The types of entities are obtained by using ClueWeb09/12 [9] annotated with Freebase [4] entities. By scanning the ClueWeb09/12 corpus with patterns (e.g. "answer" is a "type"), types from the corpus can be obtained for disambiguation. After obtaining

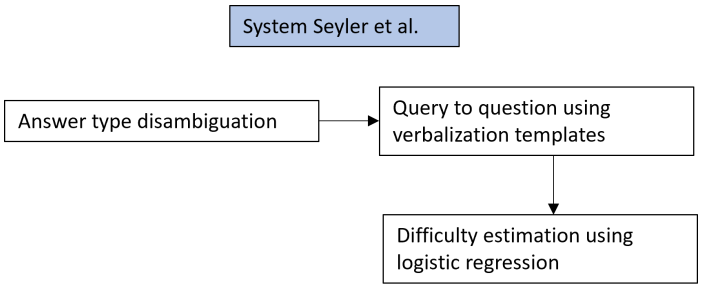

FIGURE 3.7: System of Seyler et al. [23] visualized

these types of the corpus, the answer entity type can be disambiguated by matching the entity types of the corpus with the types found in Freebase.

In order to transform the query into a question, the system uses hand crafted verbalization templates and lexicons. These verbalization templates have different rules on how to transform a query triple to a natural language question depending on its kind. An example of such verbalization in the paper is the query `AlPacino actedIn ?x` with as verbalization *Al Pacino appeared in*.

Seyler et al. [23] estimate difficulty using binary logistic regression. To train this model, a filtered Jeopardy! quiz-game dataset with only entities that Yago2s can capture is used. The input to this model is the query and its answer entity, and its output is easy or hard. The logistic regression uses the following features: entity popularity, entity super type, Jaccard similarity and answer type.

For evaluating the difficulty classifier, the logistic regression classifier using all features is compared to logistic regression classifiers using only subsets of all features. Results showed that using all features achieved the highest accuracy of 66.4%. An increase of 0.6% compared to the best performing baseline. Figure 3.7 shows the system of Seyler et al. [23] with its most important properties.

DB-quiz: a DBpedia-backed Knowedge Game

It is not clear how questions are generated in this paper, but Mynarz and Zeman [18] found out that the answer success rate to a question is correlated with the number of incoming links of the Wikipedia page of the question its topic. Another finding was that the Wikipedia page view count is even more strongly correlated to answer success rate compared to the incoming links.

Automated Template Generation for QA over KG

Abujabal et al. [1] presented QUINT, an automatic template generator. QUINT utilizes ClueWeb09-FACC1 corpus [9] annotated with Freebase [4] entities to create two lexicons: a predicate lexicon and a type lexicon. By going through the corpus with patterns, a weighted mapping is obtained. For example, for the predicate lexicon, the pattern $e_1\ r\ e_2$ is used. When encountering a phrase where there are two entities with a span in between, that span is mapped to the predicate connecting

these two entities. Since r does not always capture the predicate between the two entities, a weighting is assigned depending how many times it was encountered in the corpus.

Starting with the query (subgraph) construction, QUINT first uses an entity recognizer and disambiguator [30] to mark Freebase entities in the question. For all recognized entities, QUINT checks all paths of length one to connect all these entities. It is also required to constrain the type of the answer entity. For the answer entity, QUINT consults Freebase for the possible types.

After obtaining the query, the next step is to align tokens of QA to concepts of the KG. Aligning maps tokens to KG concepts and disambiguates the types of the answer entity. Using the information of the two lexicons, an integer linear programming problem is solved to determine the mapping and disambiguation. Placeholders are then introduced in the QA and the query, with as result a template that can be filled in with new information.

As evaluation, QUINT tried to extract templates from two datasets, WebQuestions [2] and Free917. For 3778 QAs in WebQuestions, QUINT produced 1296 distinct templates. For the 641 Free917 QAs, QUINT found 284 distinct templates.

3.2 Summary

We can see from Table 3.1 that automatic template creation is not common, with only systems of Seyler et al. [23] and Abujabal et al. [1] having a automatic template creating component. Both works use ClueWeb dataset to obtain entity annotations. Another property that stands out is difficulty estimation. MCQs systems that have difficulty estimation all incorporate distractor difficulty with only Gorgojo [27] addressing question difficulty.

	Automatic templates	General KB	Open QG	MCQs	Difficulty est.
Sherlock [16] & [15]		x		x	x
Faizan et al. [8]		x		x	x
Gorgojo [27]		x		x	x
Raynaud et al. [21]		x		x	
Chaudhri et al. [7]			x		
Song and Zhao [24]			x		
Seyler et al. [23]	x	x	x	x	x
Mynarz and Zeman [18]		x			x
Abujabal et al. [1]	x	x	x		

Table 3.1: Properties of each template-based QG paper summarized

Chapter 4

Template Extraction

Templates are required to generate new questions from examples. The template extractor expects a question with its corresponding answer as input. The output is all the information required to expand the template into new questions. Throughout the process, two examples (Table 4.1) are used to illustrate this process. The first one is made up where China is intentionally written in lowercase to illustrate pre-processing and the second one is from TriviaQA dataset [12].

4.1 Entities

The first step in template extraction is to identify which entities the entity linker should extract. There is a pre-processing step for correcting certain entities for the entity linker and an identification step for extracting relevant entities.

The inspiration to use entity linking for template creation comes from the work of Faizan et al. [8]. Faizan et al. used entity linking to mark DBpedia entities and DBpedia to obtain distractors, whereas we use entity linking as an exploration tool to mark entities for later use in Wikidata. The choice for Wikidata is because Wikidata queries are easy to compose and the relations in Wikidata are straightforward.

4.1.1 Pre-processing

In the pre-processing step, lowercase entities that are supposed to capitalized get capitalized. The tagger used for this pre-processing task is spaCy's English medium pipeline NER tagger [11]. Entities get capitalized if it belongs to one of the following classes: `GPE`, `PERSON`, `NORP`, `ORG`, `LANGUAGE`, `LOC`, `EVENT`, `PRODUCT`, `WORK OF ART`. The pre-processing step is only useful in case that the entity linker cannot detect

1. What is the capital of china? Beijing
2. Marilyn Monroe starred in The Seven Year Itch. Who directed it? Billy Wilder

TABLE 4.1: QA examples for illustrating template extraction process, with China intentionally written in lower case. Second question is taken from TriviaQA [12]

1. What is the capital of China? Beijing
2. Marilyn Monroe starred in The Seven Year Itch. Who directed it? Billy Wilder

Table 4.2: QA examples after pre-processing

lowercase entities that are supposed to be capitalized. Its effectiveness is dependent on the input QA and the entity linker's ability to detect lowercase entities that are generally capitalized.

For the two QA examples, the NER tagger recognizes the following spans belonging to one of the classes: china (`GPE`), Beijing (`GPE`), Marilyn Monroe (`PERSON`) and Billy Wilder (`PERSON`). Only the token china needs to be capitalized. Table 4.2 shows the pre-processed QA examples.

4.1.2 Identification

In this phase entities that are marked by the entity linker are transformed and saved for extraction and generating the schema. DBpedia Spotlight goes through the pre-processed QA and every entity that DBpedia Spotlight recognizes, is marked and its tag in DBpedia is returned. Immediately after marking the DBpedia entities, two checks determine whether the algorithm can extract a template from the QA. First, the algorithm checks if a DBpedia entity appears after the question mark, which is the answer entity that will be extracted to create an answer placeholder. Second, the algorithm checks if there are more than two DBpedia entities in the pre-processed QA, this is required to create at least one other placeholder in the question (question placeholder) along the answer placeholder.

DBpedia Spotlight is the entity linker used in this thesis for linking entities to DBpedia resources, but relation identification (Subsection 4.2.1) and schema construction (Chapter 5) both use Wikidata. Using Wikidata as pimary KG requires an additional transformation in the identification step to obtain the corresponding Wikidata source from DBpedia. In each DBpedia resource there is mention of its Wikidata equivalent. These equivalent tags are obtained using the following SPARQL query:

```
SELECT distinct ?same
WHERE {dbr:DBpediaTag owl:sameAs ?same}
```

Listing 4.1: SPARQL query to find similar Wikidata Q-tags

The query response contains all entities that satisfy `owl:sameAs`. Entities are considered Wikidata equivalents if they contain the URL of Wikidata. Additionally, one DBpedia entity can have multiple Wikidata sources. For example, when querying DBpedia for the Wikidata resources of Beijing, it has two Q-tags: `Q7334692` and `Q956`. The first Q-tag refers to the historical Beijing and the second Q-tag refers to the current day Beijing.

The identified entities using DBpedia Spotlight for our two examples are shown below.

1. **Pre-processed QA:** *What is the capital of China? Beijing*
   Identified entities and their Wikidata Q-tags: China `[Q127864, Q148, Q20233549, Q27769879, Q29520, Q692303, Q942154]` and Beijing `[Q7334692, Q956]`

2. **Pre-processed QA:** *Marilyn Monroe starred in The Seven Year Itch. Who directed it? Billy Wilde*r
   Identified entities and their Wikidata Q-tags: Marilyn Monroe `[Q4616]`, The Seven Year Itch `[Q290679]` and Billy Wilder `[Q51547]`

4.2 Relation

Extracting a relation between the detected entities is essential for generating new questions. There is no way to generate new entities to fill in the placeholders without a valid relation. The extracted relation should capture the semantic meaning of the template as best as possible.

The following subsections explain the relation identification and disambiguation. The second illustrating QA example is a QA where a 3-slot template can be extracted, this extraction type is explained in subsection 3-slot templates 4.3.1.

4.2.1 Identification

The relation identification phase checks whether a direct relation exists between the Wikidata tags of the answer and the question. At this point it is not clear if an entity has multiple Q-tags, which Q-tag is correct among the multiple Q-tags, thus the identification process checks every combination between the question tags and answer tags. The described way of checking direct relations automatically disambiguate the Q-tags, because it is unlikely that two incorrect tags would have a relation between the two. Once a direct relation is determined, other Q-tags are discarded as potential tags that could represent the entity.

The existence of direct relations is verified with the following Wikidata SPARQL query:

```
SELECT ?propLabel ?relationLabel
WHERE {
    wd:id_1 ?relation wd:id_2.
    ?prop wikibase:directClaim ?relation.
    ?prop rdfs:label ?propLabel.
    filter(lang(?propLabel) = "en").
    SERVICE wikibase:label
    {bd:serviceParam wikibase:language "en"}
```

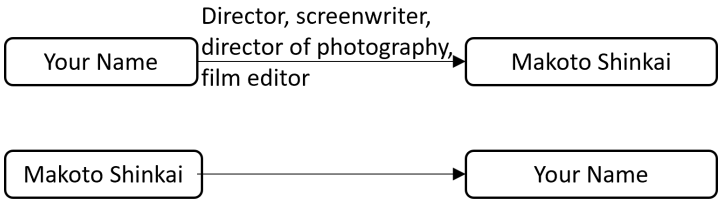

Figure 4.1: Relations are not commutative

```
}
```

Listing 4.2: SPARQL query to find direct relation between entities

The query returns the property tags and their corresponding string representations (for relation disambiguation). In case there is no relation between any combination of question Q-tag and answer Q-tag, the QA is considered as unable to extract templates from.

We made the choice to not consider multi-hop relations. Multi-hop relations are relations that connect entities with at least one other entity in between. If multi-hop relations are considered, relation disambiguation gets a lot more complicated. Even one intermediate entity may result in many relation combinations. Take for example the two entities: Chengdu and China. Although there is a direct relation in Wikidata between Chengdu (`Q30002`) and China (`Q148`), let us assume that is not the case. Chengdu is the city capital of the province Sichuan and Sichuan is a province in China. If we want to capture this 2-hop relation as capital followed by country, the query (Listing 4.3) returns 31 relations to choose from. Only 1 of the 31 possible combinations is correct.

```
SELECT ?prop ?prop2
WHERE { wd:Q30002 ?prop ?intermediate.
        ?intermediate ?prop2 wd:Q148.
        SERVICE wikibase:label
        {bd:serviceParam wikibase:language "en"} }
```

Listing 4.3: SPARQL query one intermediate hop illustration entities

Relations in Wikidata are not commutative, meaning there is not always a relation in the opposite direction. For example, there are directed edges starting from the movie Your Name (`Q21697406`) and ending at Makoto Shinkai (`Q335080`). In the opposite direction there is none. Figure 4.1 visualizes this example. The non-commutative property of Wikidata requires the above query in Listing 4.2 to be executed twice. One execution with question tag as `id_1` and the answer tag as `id_2` and another execution with the answer tag as `id_1` and question tag as `id_2`.

At the end of the identification phase, a template is generated with two placeholders.

For the China-Beijing example the algorithm checks the following combinations:

1. `Q127864` - `Q7334692`: no direct relation

2. `Q127864` - `Q956`: no direct relation

3. `Q148` - `Q7334692`: no direct relation

4. `Q148` - `Q956`: two relations found in direction `Q148` → `Q956`: contains administrative territorial entity (`P150`) and capital (`P36`). Direct relation determined, do not check the other combinations.

The result is a template with placeholders _Q_ and _A_: *What is the capital of _Q_? _A_* with above mentioned direct relations.

4.2.2 Disambiguation

After obtaining the direct relations from the identification phase, it is still required to select the relation that captures the semantic meaning of the template the best. To achieve this, the algorithm makes a semantic comparison between the keywords in the template and the relations. The relation that is semantically closest to one of the keywords is chosen as the relation that captures the meaning of the template and is used for the schema (Chapter 5).

SpaCy's English medium pipeline POS tagger tags the keywords in the QA. We made the choice to use tokens with POS tags `NOUN` and `VERB` as keywords, because templates often contain a noun or verb that captures the semantic meaning of itself. Additionally, spaCy's English medium pipeline offers a feature to transform spans to word vectors. These word vectors incorporate semantic information of the span within them and make it possible for a semantic similarity comparison.
The disambiguation process goes as follows:

1. Extract keywords from the template

2. Check if the string representation of one of the relations is in the QA. If only one relation is in the QA, then this is the relation that captures the semantic meaning.

3. Make pairwise semantic comparison between all found keyword word vectors and relation word vectors. Select the relation which has the highest similarity score to one of the keywords. This relation is the one that captures the semantic meaning of the template. If no keywords were found in the first stage, the word vector of the template substitutes the keyword word vector.

The template *What is the capital of _Q_? _A_* has two relations to choose from: contains administrative territorial entity (`P150`) and capital (`P36`). In this case, the string representation capital is in the template, resulting in capital (`P36`) as direct relation that captures the semantic meaning of the template.

4.3 Multiple Entities in Question

When there are multiple entities in the question, the relation extraction process (Section 4.2) is executed for each QA entity pair. After the relation extraction process, the algorithm checks to turn the 2-slot template into a 3-slot template.

The relation extraction process executes twice for our second QA example. Once for checking direct relations between Marilyn Monroe and Billy Wilder and the second time between The Seven Year Itch and Billy Wilder. There are only direct relations between The Seven Year Itch and Billy Wilder, after disambiguation only director (`P57`) remains. The resulting (intermediate) template is:
*Marilyn Monroe starred in _Q_. Who directed it? _A_* with relation director (`P57`).

4.3.1 3-Slot Templates

Some templates contain an additional constraint that must be extracted in order to create a valid template. The check whether a third entity extraction is required happens right after the relation identification phase (Subsection 4.2.1). The algorithm can only extract a third entity if there is at least another linked entity left in the question. The algorithm checks if there is a direct relation between the leftover question entities present in the question and the already extracted question or answer entity. In case there is a direct relation between one of the other question entities and the extracted question or answer entity, this other question entity is also extracted from the template and the relation is saved as a constraint relation for the schema.

The template *Marilyn Monroe starred in _Q_. Who directed it? _A_* still needs Marilyn Monroe to be extracted to form a template. There is a direct relation cast member (`P161`) for The Seven Year Itch (`Q290679`) → Marilyn Monroe (`Q4616`). This results in final template: *_C_ starred in _Q_. Who directed it? _A_.* with direct relation director (`P57`) and constraint relation cast member (`P161`).

4.4 Conclusion Template Extraction

To summarize this chapter, the template extraction algorithm performs identification and disambiguation for entities and relations. The extraction algorithm works best when DBpedia Spotlight can identify all entities and when there are not too many entities identified. Chapter 7 performs a more thoroughly analysis on three different QA datasets. A shortcoming is that the template extraction algorithm only checks for direct relations, this approach lowers the probability of extracting a template from a QA.

Chapter 5

Schema Construction

This chapter shows how schemas are constructed. The templates generated from the examples in Chapter 4 are used as illustration.

1. **Template:** *What is the capital of _Q_? _A_*
   Extracted entities: China (`Q148`) and Beijing (`Q956`)
   Primary relation: China → Beijing, capital (`P36`)
   Constraint relation: None

2. **Template:** *_C_ starred in _Q_. Who directed it? _A_*
   Extracted entities: Marilyn Monroe (`Q4616`), The Seven Year Itch (`Q290679`) and Billy Wilder (`Q51547`)
   Primary relation: The Seven Year Itch → Billy Wilder, director (`P57`)
   Constraint relation: The Seven Year Itch → Marilyn Monroe, cast member (`P161`)

5.1 2-Slot Templates Schema

The schema for generating new entities for 2-slot templates corresponds to the following Wikidata query:

```
SELECT DISTINCT ?newFirstLabel ?newSecondLabel
?typeFirstLabel ?typeSecondLabel
WHERE {
    wd:fromQTag wdt:P31 ?typeFirst.
    wd:toQTag wdt:P31 ?typeSecond.
    ?newFirst wdt:P31 ?typeFirst.
    ?newSecond wdt:P31 ?typeSecond.
    ?newFirst wdt:directRelation ?newSecond.
    SERVICE wikibase:label
    {bd:serviceParam wikibase:language "en"}
}
```

Listing 5.1: SPARQL query find two new entities for template

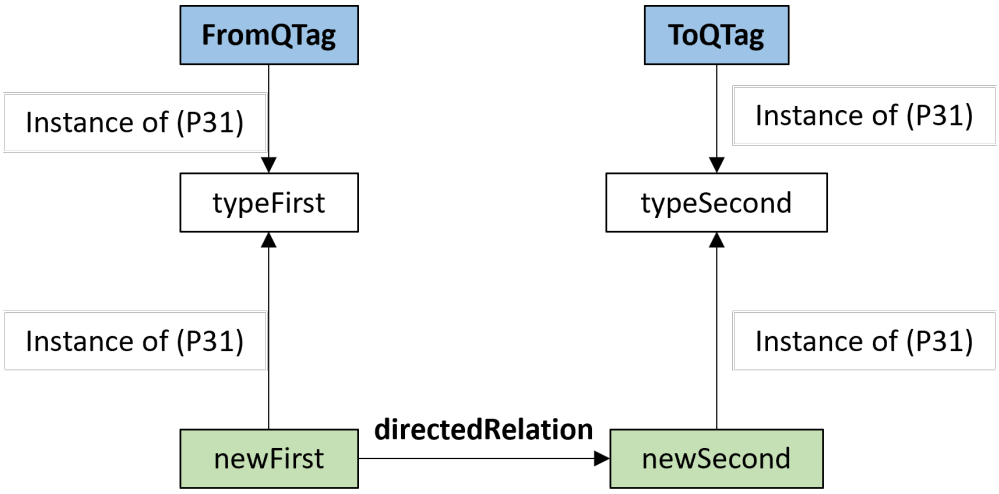

Figure 5.1: Schema for filling 2-slot templates

The above query still requires three parameters: `fromQTag`, `toQTag` and `directRelation`. These parameters correspond to `fromQTag` → `toQTag` and the primary relation connecting these two Q-tags. This query returns new entities that satisfy the primary relation and the types of the extracted entities. Figure 5.1 shows the schema with the annotations of the query.

Although the schema always constrains on the type of the generated entities, in theory, this is not required when the direct relation is only valid for specific types. An example of this is the relation family name (`P734`). The family name relation is only valid for individuals, thus constraining on the type is not required. When the algorithm deals with general relations, the schema must constrain on the types. Such a general relation is genre (`P136`), this relation specifies the genre of multiple kinds of creative work. Take for example the QA: *What is the genre of the anime Shirobako? comedy drama.* A sensible template of this QA is *What is the genre of the anime _Q_? _A_* with as direct relation genre (`P136`). If the schema does not constrain on the types of the entities, the query result will also contain pairs of movies and their genres, thus the filled in template does not make sense because not all movies are anime.

Executing this query with parameters `Q148` → `Q956` and `P36` as `directRelation` returns entity pairs of countries with their capitals. These pairs replace the _Q_ and _A_ placeholders to generate new questions. An expanded template is: *What is the capital of Belgium? City of Brussels.*

5.2 3-Slot Templates Schema

The 3-slot schema is similar to the 2-slot template schema. The query is the same with the addition of generating one extra entity depending to which entity the

constraint relation is defined.

```
SELECT DISTINCT ?newFirstLabel ?newSecondLabel
?newThirdLabel
WHERE {
    wd:fromQTag wdt:P31 ?typeFirst.
    wd:toQTag wdt:P31 ?typeSecond.
    wd:constraintQTag wdt:P31 ?typeThird.
    ?newFirst wdt:P31 ?typeFirst.
    ?newSecond wdt:P31 ?typeSecond.
    ?newThird wdt:P31 ?typeThird.
    ?newFirst wdt:directRelation ?newSecond.
    ?newFirst/Second wdt:constraintRelation ?newThird.
    SERVICE wikibase:label
    {bd:serviceParam wikibase:language "en"}
    }
```

Listing 5.2: SPARQL query to find three new entities for template

The additional parameters required compared to the 2-slot schema are `constraintQTag` and `constraintRelation`, this is the third entity and the constraint relation to which the third entity is linked to the template. Figure 5.2 shows the schema with the annotations of the query.

For the 3-slot example, the parameters are `Q290679` → `Q51547` with `directRelation` = `P57` and `constraintRelation` = `P161`. This query returns respectively films, the director of that film, and a cast member of that film for the _Q_, _A_ and _C_ placeholders. An expanded template is: *Daniel Day-Lewis starred in Lincoln. Who directed it? Steven Spielberg.*

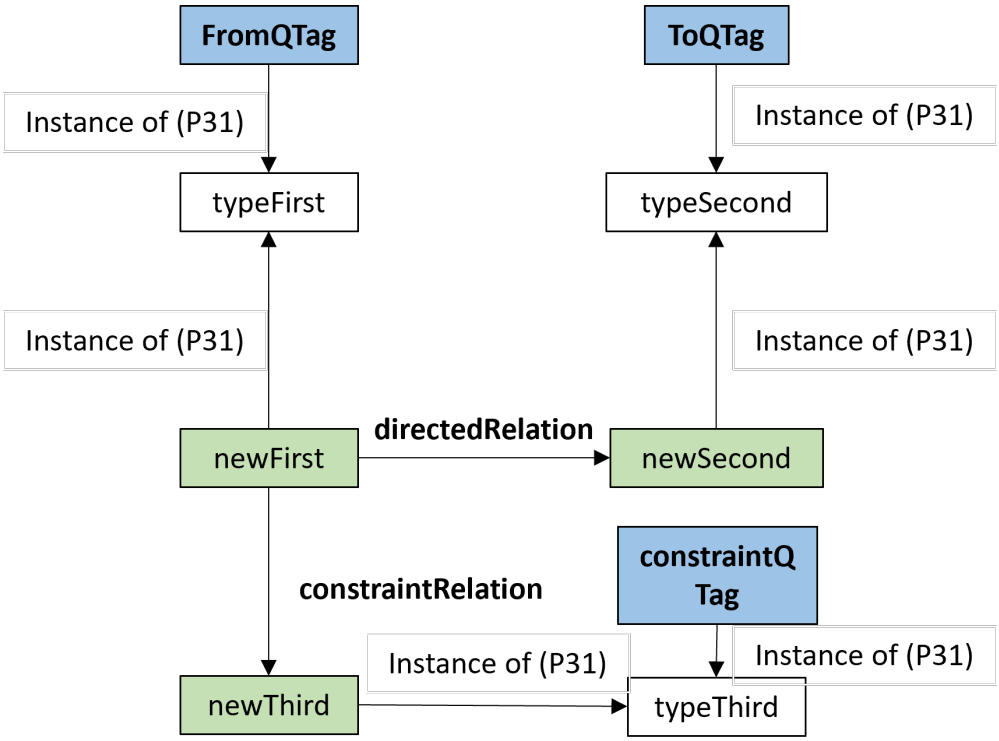

Figure 5.2: Schema for filling 3-slot templates

Chapter 6

Difficulty Ranking

Generated questions should have a difficulty estimation. Not only is a difficulty estimation useful for selecting questions that suit the audience of a trivia quiz, but can also be beneficial for selecting questions for educational assessments. Assigning an absolute difficulty category (easy or hard) to a generated question does not tell us much if there is no difficulty reference, so it is more useful to rank the generated questions based on difficulty compared to each other. This leads to a learning to rank problem for the generated questions. The chosen approach in this thesis for the learning to rank problem is pointwise ranking.

We only look at the difficulty ranking of generated questions for questions generated from 2-slot templates. This chapter does not handle difficulty ranking for questions generated from 3-slot templates.

6.1 Features

We extract features from the two extracted entities that create the slots in the template. The classifier uses features to estimate the difficulty of the question. The features are:

- Google Trends [10] global mean popularity during time period 2017/03/10-2022/03/10 of both entities
- The number of distinct incoming Wikidata properties of both entities
- Jaccard similarity: The number of entities that have a direct incoming link to the question and answer entity

The Google Trends mean popularity feature draws inspiration from the works of Faizan et al. [8] and Seyler et al. [23]. The system of Faizan et al [8] uses the well-knowness of SPO triples as difficulty estimation for their Jeopardy style QG. The difficulty logistic regression classifier of Seyler et al. [23] uses a feature that is similar to entity popularity. We made the choice of using Google Trends as a popularity feature, since the numbers on Google Trends directly relate to popularity. The number of incoming Wikidata statement is inspired by systems of Gorgojo

	4g	linux	reddit	instagram
0	5	10	35	134

Figure 6.1: Absolute scale Google Trends

[27] and the finding of Mynarz and Zeman [18]. The system of Gorgojo [27] uses a combination of incoming and outgoing of links DBpedia to estimate difficulty, and Myanrz and Zeman [18] found that the difficulty correlates with the number of incoming links of the question topic's Wikipedia page. We made the choice to use number of distinct incoming properties instead of number of incoming links, because the number of incoming links might give a false impression. Take for example the manga of Naruto (`Q26971382`), it has 206 incoming links, but many links are from characters of the manga. Both Google Trends mean and number of distinct incoming Wikidata properties show how well-known the entities are. Assuming that the more well-known an entity is, the easier the question. The Jaccard similarity tells how much the entities are related to each other and is also inspired by Seyler et al. [23], where their system uses it as a feature for their difficulty logistic regression classifier.

```
SELECT (COUNT(DISTINCT(?prop)) as ?count)
WHERE {
?item ?prop wd:QTag/ATag.}
```

Listing 6.1: SPARQL query to find number of distinct incoming properties

```
SELECT (COUNT(?item) as ?count)
WHERE {
?item ?prop1 wd:QTag.
?item ?prop2 wd:ATag.}
```

Listing 6.2: SPARQL query find the Jaccard similarity

The mean popularity of Google Trends requires a transformation before it is used as a feature value. This is because Google Trends outputs the popularity of items on a relative scale depending on the other item(s) it is paired up with in the trend search. The method of going from relative popularity to an absolute popularity is to find a range of items that have a stable popularity.

The four threshold items used for conversion are: *4g*, *linux*, *reddit* and *instagram*. We found these four items after trial-and-error. All these search terms have a popularity with low variance throughout the time period 2017/03/10-2022/03/10, and define different thresholds of popularity. *Linux* is about 2 times more popular than *4g*, *reddit* is about 3.5 times more popular than *linux*, and *instagram* is about 3.8 times more popular than *reddit*. With these ratios, it is possible to map the relative scale to an absolute scale (Figure 6.1). We made the choice to map *4g* to value 5, every other threshold value follows from the ratios. For each entity, the feature extractor determines between which threshold items the entity lies before mapping it to the absolute scale to have a more fine-grained popularity score.

6.2 Dataset

A machine learning classifier requires a dataset for training to output some relevant label that is comparable to difficulty. The data in this dataset should also contain two entities that appear in Wikidata for feature extraction. At the time of writing, there is no conventional QA dataset that contains difficulty labels. For this thesis, we collected data through a quiz website called Sporcle [25]. Sporcle groups quiz questions into a specific theme quiz and each question has a percentage of participants that were able to correctly answer the question. This thesis uses that percentage as the label for measuring difficulty.

In total 186 questions were collected to train the classifier. All of these questions were open, simply structured questions containing only two entities, furthermore only quizzes with sufficient participants were considered to have an accurate representation of the difficulty. One data example consists of the feature vector extracted from the two entities and the percentage as label.

6.3 Classifier

After determining the features and gathering the data, a machine learning classifier trains on the data to output the probability that the question would be answered correctly. For choosing a classifier, we compared the performance of the classifiers using the same metrics of Section 8.1. Although the performance difference between classifiers was not significant, random forest performed the best out of all of them. It should be noted that we did not consider deep learning as a possible classifier because one of the goals of this thesis is to make an interpretable system.

The numerical output label of the random forest is used for pointwise ranking. Generated questions are sorted on this output label in descending order to obtain a ranking from easy questions to hard questions.

Chapter 7

Template Evaluation

This chapter focuses on evaluating the template extractor described in Chapter 4. We evaluate two important questions: (1) how often can the template extractor extract a template from a QA pair and (2) how correct are the expansions of such a extracted template, i.e. that results in a correct, new QA pair. We make no attempt at quantifying the trivia worthiness of templates since it is highly dependent on the data and the knowledge graph.

The first Section 7.1 describes the setup used for evaluation. Results of evaluation are found in the second Section 7.2.

7.1 Evaluation Setup

The template extractor requires datasets to extract templates from. For evaluation, the template extractor performs its algorithm on three datasets:

- **SimpleQuestions** [5]: QA dataset with the answer as a Freebase entity. From SimpleQuestions, only QA pairs where Wikidata could translate the answer entity to a string representation based on its Freebase tag are used for extraction. These questions are not necessarily "trivia-worthy" in the sense that some questions have multiple correct answers. From this dataset, 1800 QA pairs are used for to extract templates from.
- **WebQuestions** [2]: QA dataset where the QA pairs consistently contain knowledge base entities. 1000 QA pairs are used for template extraction.
- **TriviaQA** [12]: A more complicated QA dataset where the questions are similar to questions of trivia quizzes. 1500 TriviaQA QA pairs are used for template extraction.

Some data examples for the three datasets:

- **SimpleQuestions**: *What is a child genre of strategy game? grand strategy wargame*
  What language is the show elementary broadcast in? English
  where is the 58033 *zip code? Fort Ransom State Park*

- **WebQuestions**: *what character did john noble play in lord of the rings? Denethor II*
  who does joakim noah play for? Chicago Bulls
  where are the nfl redskins from? Washington Redskins

- **TriviaQA**: *Which radioactive substance sometimes occurs naturally in spring water? Radon*
  On the London Underground, which is the only line to connect at some point with every other line on the system? Jubilee line
  Who composed the musical theme for the Pink Panther? Henry Mancini

For each of these QA pairs, the template extraction process checks if it can extract templates. These templates can be 2-slot or 3-slot templates. QA pairs where 3-slot templates are extracted from often contain duplicate templates. For example, the QA: *What currency does the Dominican Republic use? Dominican peso* contains two templates:

1. **Template:** *What _Q_ does the _C_ use? _A_*
   Extracted entities: currency (`Q8142`), Dominican Republic (`Q786`) and Dominican peso (`Q242922`)
   Primary relation: Dominican peso → currency, instance of (`P31`)
   Constraint relation: Dominican Republic → Dominican peso, currency (`P38`)

2. **Template:** *What _C_ does the _Q_ use? _A_*
   Extracted entities: currency (`Q8142`), Dominican Republic (`Q786`) and Dominican peso (`Q242922`)
   Primary relation: Dominican Republic → Dominican peso, currency (`P38`)
   Constraint relation: Dominican peso → currency, instance of (`P31`)

These two templates are the same with the primary relation and the constraint relation switched. In the first template, the constraint relation forces the _A_ placeholder be some kind of currency and the primary relation ensures that _Q_ placeholder's new entity is currency. For the second template, now the primary relation forces _A_ to be some kind of currency again and the constraint relation fill the _C_ placeholder with currency. Both schemas result in the same entities when queried, thus we remove all but one of these duplicated templates. For each template the following information is used to evaluate the template:

- The QA where the template is extracted from.
  E.g. *What is the currency used in China? Renminbi*

- The generated template out of the QA
  E.g. *What is the currency used in _Q_? _A_*

- The types that the Wikidata SPARQL query infers in Listing 5.1 for generating new entities for the placeholders.
  E.g. (country, currency)

- The constraint relation with the corresponding entity

- An example of expansion of the template
  E.g. *What is the currency used in Belgium? Euro*

Since there are no metrics for evaluating the generated templates on correctness, human evaluators were consulted for evaluating the templates. Each template is evaluated three times by different evaluators, the templates are assigned either "correct" or "incorrect". These evaluators were asked to rate templates based on the criteria if the generated question after expansion is an objectively "correct" QA pair. With "correct" being: (1) the filled in entities make sense in the context of the template and (2) syntactically correct. The enumeration below shows examples of "incorrect" templates. Some templates could not be expanded with their corresponding schemas, because the processing time of the query exceeded the one-minute processing time limit of Wikidata. Although these templates are not filled in, the other information that is available at each template (see above) is sufficient to determine whether the filled in template would be "correct", and thus the results of these templates are also taken into account for evaluation. For each template, we take the majority vote to obtain the final label.

For a more fine-grained analysis, incorrect templates that are categorized into nine different types:

1. **Faulty relation between entities**: The relation between extracted entities does not capture the semantic meaning of the template. E.g. Template: *Where is _Q_* ***born****?* _A_
   Extracted relation: **city of occupation**

2. **Constraint present**: The template still contains a word that constrains the possible values of the placeholders, this word should have been extracted to create a three-placeholder template. Expansion of templates with this kind of incorrectness results in QA pairs that do not necessarily satisfy that constraint. E.g. Template: *Who voiced _Q_ in* ***Shirobako****?* _A_
   Extracted relation: voice actor

3. **Leftover extracted entity**: The template still contains a part of the entity which the algorithm could not fully extract from the QA pair. E.g. QA: *In which country is Grand Canyon located? The United States of America*
   Template: *In which country is _Q_ located?* ***The United States of*** *_A_*

4. **Comparatives**: The template contains a time element or comparable property (best, longest, shortest, second place...). Although this falls under the second category constraint present, this incorrectness occurs often enough to warrant its own category. E.g. Template: *Who won the* ***most*** *medals during _Q_ ? _A_*
   Extracted relation: Medal winner

5. **Slight grammatical error**: The QA where the template is extracted from is worded weirdly or the expanded template contains some grammatical mistake. E.g. QA: *What country utilizes the tomadino language? Indonesia*
   Template: *What country utilizes **the** _Q_? _A_*
   Expanded template: *What country utilizes **the English**? New Zealand*

6. **Type Inference:** At template expansion the type inference by the query does not always generate correct generated QAs. E.g. QA: *Name an artist associated with the baroque movement? Giovanni Battista Salvi Da Sassoferrato*
   Template: *Name an artist associated with the _Q_ movement? _A_*
   Extracted relation: movement
   Inferred types: architectural style and painting

7. **QA not suited**: This QA is not suited for the template extractor described in this thesis because it has more than two entities in the question. E.g. QA: *Who is the voice actor that voiced in Durarara!!, Kuroko no basket and Shingeki no Kyojin? Daisuke Ono.*

8. **Wrong entity**: Another entity should have been extracted from the question to create the template. E.g. QA: *What is the **capital** of China? Beijing*
   Template: *What is _Q_ of **China**? _A_*

For assigning the labels, we first check if the template violates the error types: 1, 7 or 8. The reason these three labels are considered first is because the other types of errors cannot be properly assessed without these three types being correct. If the template passes these three criteria, we assign one of the types of 2, 3, 4, 5 or 6.

7.2 Results

This section reviews the results of the evaluation setup. The section starts with evaluating the questions to templates ratio for the three datasets in Subsection 7.2.1, and ends with Subsection 7.2.2 that looks at the overall quality of templates and the incorrect templates.

7.2.1 Extraction Ratio

Our template extraction algorithm can only handle a subset of possible QA, meaning that the algorithm cannot extract a template from all QA pairs. The ratio of successfully extracted templates for each dataset is described in Table 7.1.

Table 7.1 shows that the template extraction performs the best with the data from WebQuestions. This is not surprising, since the QA data in WebQuestions are short questions often containing entities that are present in DBpedia. Although the number of templates extracted from TriviaQA's looks promising, this is because of the length of the questions in TriviaQA. Questions in TriviaQA are often long questions containing a lot of entities, this increases the number of entities that

Dataset	# total extracted templates	# 2-slot templates	# 3-slot templates	% successful extraction
SimpleQuestions	360 (1800 QA)	332	28	20.0%
WebQuestions	415 (1000 QA)	367	48	41.5%
TriviaQA	497 (1500 QA)	336	161	33.1%

TABLE 7.1: Metrics questions to templates

	SimpleQuestions	WebQuestions	TriviaQA
Total found templates	360	415	497
Total 2-slot templates	**332**	**367**	**336**
Correct 2-slot templates	60% (200)	43% (156)	9% (30)
Incorrect 2-slot templates	132	211	306
Total 3-slot templates	**28**	**48**	**161**
Correct 3-slot templates	57% (16)	52% (25)	6% (9)
Incorrect 3-slot templates	12	23	152

TABLE 7.2: Template evaluation distribution

DBpedia Spotlight can mark. More entities means more likely to find a direct link between the answer and question entities, thus resulting in more templates.

There are two main reasons why the template extractor could not extract a template from a QA:

- **Entity linker**: The template extraction relies on DBpedia Spotlight. The main problem is that DBpedia Spotlight does not recognize an entity that is supposed to be extracted. Sometimes it cannot recognize an entity because of the lack of context, since DBpedia Spotlight performs better on (long) text rather than single sentences. Other times, either the entity is not present in DBpedia or the confidence parameter is too high. Playing with the confidence parameter of DBpedia Spotlight has a small impact on the number of entities recognized, resulting in 1 or 2 more templates per 100 QA. Setting the confidence parameter lower for entity search might seem the way to go, but this is at risk of finding direct links between entities that DBpedia Spotlight should not extract.
- **No direct relation**: The extracted entities might not have a direct relation between them, either because of Wikidata does not have the expected relation or the relation exists as a multi-hop relation. The template extraction only looks at single hops between two entities, Wikidata might connect the entities might with other entities in between.

7.2.2 Template Quality

We obtain two results after performing the human evaluation described in Section 7.1. Table 7.2 shows the ratio of correct-incorrect extracted templates and Table 7.3

shows a further breakdown to which type each incorrect template belongs to.

Correct Templates Analysis

There is a high rate of correct templates extracted from SimpleQuestions and WebQuestions. For both datasets, around 25-30% of the correct 2-slots templates belong to two relations. In case of SimpleQuestions, 30 templates ask about place of birth (`P19`) and 24 about country of citizenship (`P27`). For WebQuestions, the top two relations are: educated at (`P69`) containing 22 templates and member of sports team (`P54`) containing 20 templates. The correct 2-slot templates of SimpleQuestions contain 52 unique direct relations, and 2-slot WebQuestions templates contain 49 unique direct relations. This amount depends on the questions in the dataset, but this is a good indication that the template extractor can extract a wide variety of templates. Template extraction performs very poorly on TriviaQA, out of the few correct templates, the correct 2-slot templates span over 19 unique direct relations. It is not surprising to see that the correct 2-slot templates are all from short questions without too many words, because the template extraction is more likely to extract incorrect templates from long questions.

Overall, the correct 3-slot templates are passable. There are some interesting templates, but all templates that contain the instance of relation `(P31)` (either as primary relation or constraint relation) are 2-slot templates under disguise. The generated entity from the instance of relation that replaces the _C_ or _Q_ placeholder is the same entity as the extract entity from the QA, because the other extracted relation forces the entity where instance of is checked against to be an entity of a certain type. The duplicate templates example in Section 7.1 is such an example where the 3-slot template is a 2-slot template under disguise. In this example the currency relation forces the _A_ entity to be a monetary unit, so the third entity is always currency.

Incorrect Templates Analysis

We classify the incorrect templates with the error type classification labels from Section 7.1. Without counting type 7 (QA not suited) errors, type 1 (faulty relation) and 2 (constraint present) errors contribute to the majority of the errors. Type 1 error indicates that the extracted relation does not capture the template well enough to be a valid template. This shows that Wikidata does not contain many unconventional relations (e.g. favorite food), making it less versatile to use for template extraction. A possible solution is to use other KBs as main KB, such as DBpedia [14] or YAGO [20]. Another solution that should reduce the amount of type 1 errors is to build a trivia KB containing these unconventional relations for the relation identification phase (Subsection 4.2.1) in template extraction (Chapter 4). With this solution, the relation identification phase checks for relations in Wikidata and the custom build KB.

2-slot templates that the template extractor could not make into 3-slot templates are categorized as type 2 errors. Type 2 errors are somewhat dependent on both

FIGURE 7.1: Wikidata [28] qualifiers for the statement: `Q16069692` (Luc Sels) - `P39` (position held) - `Q212071` (rector), image taken from the Wikidata page of Luc Sels [29]

Wikidata and DBpedia Spotlight. There are three reasons why type 2 errors occur: DBpedia Spotlight does not recognize the entity it should have, Wikidata does not contain a direct relation to the constraint entity and one of the other extracted entities, or the entity does not exist in Wikidata. The amount of type 2 errors indicates that the used tools are not always optimal for generating templates when the questions are a bit more complicated. The solution for reducing type 2 errors is the same as the solutions proposed for reducing type 1 errors. With a custom KB, it is also possible to train an entity linker. This allows the entity linker to recognize the constraint entity and should allow the template extractor to extract the extra slot.

The entity linker is responsible for the error types: 3 (leftover entity in template), 5 (grammatical error in template) and 8 (extraction of wrong entity). If the entity linker is a bit more flexible in its tagging, error types 3 and 5 can be avoided. Type 8 error occurs because the entity linker could not recognize the correct entity to extract from the original QA. Changing the confidence parameter could solve this problem. It is also possible that DBpedia does not contain the entity that should be extracted.

Type 4 errors are errors related to templates that still have the comparative or time element. Trivia questions are often questions containing a comparative or a time element. The only way to generate a template from these kinds of questions is to take the comparative or time element token out of the question. In theory, they can be extracted using Wikidata's additional information in a statement. Wikidata sometimes provides additional information along statements, called qualifiers. Figure 7.1 shows the qualifiers for the statement: `Q16069692` (Luc Sels) - `P39` (position held) - `Q212071` (rector). One of the qualifiers shows for example the starting time for this position. It is possible to query for all the rectors of KU Leuven and sort them based on their starting time. This sorted list gives information about the chronological order of KU Leuven rectors and can be used for dealing questions questions like *Who is the second rector of KU Leuven?* However, this approach for extracting comparatives or time elements from the QAs does require additional rules to be implemented for extraction, another issue is that Wikidata does not provide qualifiers to all statements.

	SimpleQuestions	WebQuestions	TriviaQA	total
Total incorrect	**144**	**234**	**458**	**836**
Type 1	18% (26)	37% (86)	19% (85)	24% (197)
Type 2	20% (29)	24% (56)	13% (60)	17%(145)
Type 3	8% (12)	11% (26)	3% (13)	6% (51)
Type 4	1% (1)	12% (28)	9% (40)	8% (69)
Type 5	7% (10)	4% (9)	1% (3)	3% (22)
Type 6	2% (3)	1% (2)	1% (6)	1% (11)
Type 7	15% (21)	2% (4)	53% (244)	32% (269)
Type 8	29% (42)	10% (23)	2% (7)	9% (72)

Table 7.3: Templates error distribution

Chapter 8

Difficulty Ranking Evaluation

In this section we evaluate the difficulty ranking of the generated questions. In the first section, the random forest classifier is evaluated on two metrics: mean squared error (MSE) and minimum number of inserts required to obtain ground truth ordering. A human evaluation is also performed to measure usefulness of using a difficulty classifier versus just using one of the features as a difficulty heuristic. Section 8.2 describes the evaluation setup for human evaluation and the obtained results.

8.1 Evaluation Metrics

For the two following evaluations, we split the gathered Sporcle [25] dataset into two sets: a training set containing 70% of the data and a test set containing 30% of the data. The labels are percentages of people that got the answer correctly and these labels can be seen as the probability of getting the answer correct. We fit the training set to the random forest implementation of Sklearn's API [6], with a random seeding of three and a max depth of five. These two parameters do not mean much and is only for us to obtain a deterministic random forest across different calls for consistency. All other parameters are the default parameters of Sklearn.

8.1.1 Mean Squared Error

Measuring the MSE of a classifier compared to a baseline is one of the ways to check if the classifier performs well. The random forest is put against a baseline that always predicts the mean percentage of the labels. For our dataset, the baseline prediction is 63.01. The MSE with the test set can be found in Table 8.1. This solidifies that the random forest is better at predicting the numeric value compared to the baseline.

Although MSE can be used to measure the quality of the classifier for pointwise ranking, it does not directly indicate the performance for ranking. The squared root of the MSE is 17.8, which is very high since the numerical labels in the dataset are between 11.9 and 97.8. In a classic regression setting, this MSE would indicate that

	MSE
Baseline (predicting mean y)	616
Random forest classifier	320

Table 8.1: MSE of baseline and random forest classifier

the classifier is too inaccurate. In a ranking setting however, the prediction can all be off with by a constant c and the ranking would be still be 100% correct.

8.1.2 Number of Inserts

Instead of using MSE to measure the performance. It is more informative to evaluate the ordering of the classifier compared to the ordering made by the ground truth. The choice for evaluation is the minimum number of inserts required to obtain the ground truth ordering. The minimum number of swaps is not used as an evaluation metric because it does not indicate the ordering performance. The following example shows the difference:

```
Ground truth ordering = [a b c d e]
  Pointwise ordering = [e a b c d]
```

min. number of inserts: 1 vs min. number of swaps: 4

For this evaluation, the ranking obtained from the classifier is used to compare to three heuristic models. Each heuristic model is based on one of the features for training the classifier:

- **Google Trends [10] heuristic**: This heuristic model approximates the difficulty by adding the popularity of the _Q_ entity and _A_ entity and ranks the sums in a descending order. The idea is that the more popular the entities are, the more likely that a participant can answer the question correctly.
- **Wikidata distinct incoming properties heuristic**: The second heuristic model is ranking based on the number of distinct incoming properties. The heuristic model adds the number of distinct incoming properties of _Q_ entity and _A_ entity and ranks the sums in a descending order. Entities with more incoming properties should be more well-known compared to entities with fewer incoming properties, thus a participant has a higher probability to know the answer.
- **Jaccard similarity heuristic**: The last heuristic model ranks the questions based on the Jaccard similarity. A higher Jaccard similarity should make the question easier, since the two entities are more related.

With these three heuristic models and the random forest classifier, below are the steps for setting up the evaluation:

	Classifier	Google base	Links base	Jaccard base
# avg inserts	1.43	1.82	1.62	1.65

TABLE 8.2: Number of inserts required to obtain ground truth ordering (5 items)

1. Split data randomly in a train and test set with a ratio of 7:3.
2. Build the random forests classifier with the training data and obtain the predicted numerical labels of the test set.
3. Randomly sample five items from the test set without replacement until the test set contains less than five items. We do this because the test set contains a limited number of elements and we want to avoid sampling items that appear in multiple samples.
4. Take one series of these five items, sort them on descending order based on the true labels, this is the ground truth ordering. Check what the minimum number of insert is to obtain the ground truth ordering from the random forest classifier ordering (obtained by sorting the output labels in descending order). Also check the minimum number of inserts required to obtain the ground truth ordering with the three heuristic models predictions.
5. Repeat step 4 until all series are checked. Average the minimum number of inserts required for each model.
6. Repeat step 1 through 4 500 times and average the result.

Table 8.2 shows the results of the evaluation setup. The pointwise ranking approach using random forests does indeed perform slightly better compared to the heuristic models for ranking. The performance increase is only marginally of 0.2 and 0.4. These numbers do not tell us much because the minimum number of inserts that can be performed is 1. Another problem is whether the collected data of Sporcle is general enough to use it to classify the difficulty. We look at these two problems in the next section, where we perform a human evaluation to see if the classifier is indeed useful.

8.2 Human Evaluation

In this section, we perform one last human evaluation to determine the usefulness of the classifier with the main question being: Do humans prefer the ranking made by the classifier over the rankings made by the heuristic models?

8.2.1 Evaluation Setup

To determine the preference for each ranking model, we used a set of templates filled in with their corresponding Wikidata entities (Table 8.3). From these generated

Templates	Kind of entities to fill in
What kind of money to take to _Q_ ? _A_	Country & monetary unit
_Q_ is in which country? _A_	Airport & country
In which country is the region _Q_ ? _A_	Region & country
What is the official language of _Q_? _A_	Country & language
Which country has the internet domain _Q_? _A_	Internet domain & country
What country is _Q_ part of ? _A_	Island & country
_Q_ is located in what country? _A_	Lake & country
What is the capital of _Q_? _A_	Country & capital
Who is the director of _Q_? _A_	Film & director
Which company developed _Q_? _A_	Game & developer

Table 8.3: Templates for human difficulty evaluation

	Classifier	Google model	Links model	Jaccard model
# preference	80 (33%)	51 (21%)	45 (19%)	65 (27%)

Table 8.4: Preference count for the different difficulty rankings models

QAs, 5 were randomly sampled without replacement and this is repeated 50 times, resulting in 50 series of 5 QAs each. 4 rankings were generated for each series, 3 using the heuristic models described above and 1 using the random forests classifier. For each series, we presented the four rankings in a randomized fashion and without revealing the model that generated the ranking. We asked participants to mark the ranking that represents the best ranking from easy to hard difficulty. In total 4 people participated in the human ranking evaluation, totalling 200 preferences. If a ranking of preference is equal with another ranking, both of these get an increment on the preference count.

Note that the rankings showed the questions as well as the answers to those questions. This prevents participants from accidentally thinking a hard question is an easy question. For example, most people might think that the capital of Australia is Sydney. These people might classify the question: *What is the capital of Australia?* as easy, but this question is not easy as these people might think since the answer is Canberra.

8.2.2 Results

Table 8.4 shows the preference count. This shows that the classifier is useful for ranking the questions from easy to hard and that the features combined contribute to a better ranking.

Chapter 9

Conclusion

This final chapter summarizes the results, answers the research questions presented in Section 1.2 and discusses the limitations and future improvements of the system.

9.1 Discussion of Created System

We created a question generation system that is able to create automatic templates based on entity linking, and then rank the generated questions based on difficulty. The created system solves all the four problems presented in Section 1.2, maybe with the exception that the generated questions are not always syntactically and semantically correct. Syntactic correctness highly depends on the given QA where a template is extracted from and the entity linker's ability to tag entities. Semantic correctness depends on the relation extraction (one or multiple hops) and information that is present in the knowledge base.

Our contribution to template-based QG systems is creating an algorithm that generates open question templates using entity linking. In this thesis, the template extraction algorithm uses DBpedia Spotlight for highlighting the entities for extraction and the extraction itself is with the help of Wikidata. However, as mentioned in Subsection 2.4.2, spaCy offers a trainable entity linker that can be tuned to any knowledge base. By using our entity linking approach for template extraction, the template extraction algorithm can be generalized to work with other knowledge bases.

For ranking the generated questions based on difficulty, we used a pointwise learning to rank approach with random forests. Results showed that a reasonable difficulty ranker can be created from a rather small dataset by using the features described in Section 6.1. Similar features were already proven to be useful in other works. Seyler et al. [23] used Jaccard similarity as a feature for their logistic regression difficulty classifier, Gorgojo [27] showed a difficulty estimation using in- and outgoing links of DBpedia entities, and Mynarz and Zeman [18] showed that the incoming links of the Wikipedia pages are related to difficulty. Our contribution is creating a difficulty regression classifier that incorporates three features: Google Trends popularity, number of distinct incoming Wikidata statements and the Jaccard

similarity. Instead of showing the predicted label of the regression classifier, which is not that informative, we sort the predicted labels to obtain a ranking of generated questions based on difficulty.

9.2 Limitations & Future Improvements

The proposed system has some limitations. Limitations were either found during evaluation or are limitations we implemented during the creation of our proposed system.

Direct relation extraction There is no doubt that some QAs contain templates that could only be extracted by checking for multi-hop relations. The solution that is mentioned several times is to generalize the relation identification to allow multi-hops, this requires some additional rules in the algorithm to make sure that the relation disambiguation performs correctly.

Limited slots extraction The current system is limited to extracting templates that contain a maximum of three slots. Some additional rules could be added to extract templates with 3+ slots and schemas could also be created for these 3+ slot templates.

Pointwise ranking The difficulty ranking algorithm uses a pointwise approach, but learning to rank also has two other approaches: pairwise and listwise. This thesis did not study the performance difference between these three approaches since we consider it as an optimization.

Multiple-choice questions The system does not tackle generation of MCQs, although the system could be easily extended with it. There are two easy ways to generate distractors given the current system: (1) construct a schema that constrains on the type of the key or (2) use other returned entities from the schema. For estimating the difficulty of MCQs, we can extend the difficulty ranker with distractor difficulty estimators.

Difficulty ranking 3-slot templates Difficulty ranking is only looked at for questions generated from 2-slot templates. The difficulty classifier could be modified to take the third entity into account for difficulty ranking.

Lack of user interaction Originally, the idea was to allow for user interaction during the question generation process. Useful user feedback could be: selecting the templates to expand and give the user the option to review templates and correct them. Google Trends API provides the category to which a term belongs to, which can be used to categorize the templates for users to select from. The ability to correct incorrect templates could be useful to obtain more template variety, but this requires manual checks and understanding of how the system works.

Bibliography

[1] A. Abujabal, M. Riedewald, M. Yahya, and G. Weikum. Automated template generation for question answering over knowledge graphs. In *26th International World Wide Web Conference, WWW 2017*, pages 1191–1200, 2017.

[2] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.

[3] J. Biega, E. Kuzey, and F. Suchanek. Inside yago2s: a transparent information extraction architecture. pages 325–328, 05 2013.

[4] K. D. Bollacker, C. Evans, P. K. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, 2008.

[5] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[6] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[7] V. K. Chaudhri, P. Clark, A. Overholtzer, and A. Spaulding. Question generation from a knowledge base. In *EKAW*, 2014.

[8] A. Faizan and S. Lohmann. Automatic generation of multiple choice questions from slide content using linked data. In *Proceedings of the 8th International Conference on web intelligence, mining and semantics*, WIMS '18, pages 1–8. ACM, 2018.

[9] E. Gabrilovich, M. Ringgaard, and A. Subramanya. Facc1: Freebase annotation of clueweb corpora, version 1. 06 2013.

[10] Google. Google trends. https://www.google.com/trends.

[11] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020.

[12] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[13] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari. A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*, 30(1):121–204, 2020.

[14] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. 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, 01 2014.

[15] C. Lin, D. Liu, W. Pang, and E. Apeh. Automatically predicting quiz difficulty level using similarity measures. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, New York, NY, USA, 2015. Association for Computing Machinery.

[16] D. Liu and C. Lin. Sherlock: a semi-automatic quiz generation system using linked data. In *SEMWEB*, 2014.

[17] P. N. Mendes, M. Jakob, A. Garcia-Silva, and C. Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics)*, 2011.

[18] J. Mynarz and V. Zeman. Db-quiz: a dbpedia-backed knowledge game. *Proceedings of the 12th International Conference on Semantic Systems*, 2016.

[19] A. Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[20] T. Pellissier Tanon, G. Weikum, and F. Suchanek. Yago 4: A reason-able knowledge base. In A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, editors, *The Semantic Web*, pages 583–596, Cham, 2020. Springer International Publishing.

[21] T. Raynaud, J. Subercaze, and F. Laforest. Thematic question generation over knowledge bases. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 1–8. IEEE, 2018.

[22] I. V. Serban, A. García-Durán, C. Gulcehre, S. Ahn, S. Chandar, A. Courville, and Y. Bengio. Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 588–598, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[23] D. Seyler, M. Yahya, and K. Berberich. Knowledge questions from knowledge graphs. *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, Oct 2017.

[24] L. Song and L. Zhao. Question generation from a knowledge base with web exploration, 2017.

[25] Sporcle. Sporcle website. https://www.sporcle.com/.

[26] A. Thalhammer, N. Lasierra, and A. Rettinger. Linksum: Using link analysis to summarize entity data. In *ICWE*, 2016.

[27] G. Vega-gorgojo. Clover quiz: A trivia game powered by dbpedia. *Semantic Web*, 10, 06 2018.

[28] D. Vrandecic and M. Krotzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[29] Wikidata. Wikidata page of entity q16069692. https://www.wikidata.org/wiki/Q16069692. Used on 2022/05/27.

[30] Y. Yang and M.-W. Chang. S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking, 2016.

[31] R. Zhang, J. Guo, L. Chen, Y. Fan, and X. Cheng. A review on question generation from natural language text. *ACM Transactions on Information Systems*, 40:1–43, 1 2022.