Automated Generation of Customizable Recall Questions

Eran Raveh

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Arts in International Studies in Computational Linguistics

Supervisor: Prof. Dr. Detmar Meurers

September 6, 2015
Dedicated in great honour to my grandfather, Shlomo Hanns Fischer, by whom I was inspired so much, who always supported us, and who I wish could have read these words...
Towards the end of my studies, I would like to thank all of those who helped me and supported me during this time, and especially the professors and lecturers from whom it fell in my lot to learn.

My personal thanks to my supervisor, Professor Detmar Meurers, who guided me through the process of writing this thesis, gave me good advice about the next steps to be carried out — and always had at least a few more ideas for what else could be done.

Many thanks also to Professor Christopher Culy for believing in me and in my work, and for all the opportunities he has given me — directly and indirectly.

Last but not least, special thanks to Abby Huber for her very helpful linguistic editing and proofreading.

Eran Raveh
Abstract

This paper presents an end-to-end system for automatic question generation for authentic texts, using a novel approach based on customizable patterns. The goals of the system are reading-check and verification of basic understanding of a text. It detects the main terms in the texts, generates questions about them and selects appropriate distractors for them — all are based purely on the given text itself. Additionally, ranking and filtering mechanisms are implemented for choosing the best tasks. The user can modify the task patterns and create new ones for making the system focus on the desired contents in the text and generate questions best suited to the learning goals.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Outline</td>
<td>3</td>
</tr>
<tr>
<td>2 Background and Related Works</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Reading Comprehension and Recall Questions</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Automated Question Generation</td>
<td>6</td>
</tr>
<tr>
<td>3 The System — ARCQC</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Architecture Overview</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Text Analysis</td>
<td>13</td>
</tr>
<tr>
<td>3.3 Task Generation</td>
<td>16</td>
</tr>
<tr>
<td>3.3.1 Target Detection</td>
<td>16</td>
</tr>
<tr>
<td>3.3.2 Term Extraction</td>
<td>17</td>
</tr>
<tr>
<td>3.3.3 Instruction Formulation</td>
<td>22</td>
</tr>
<tr>
<td>3.3.4 Distractor Selection</td>
<td>23</td>
</tr>
<tr>
<td>3.4 Overgeneration and Ranking</td>
<td>26</td>
</tr>
<tr>
<td>3.5 Customization and Generalization</td>
<td>28</td>
</tr>
<tr>
<td>3.5.1 Pattern Structure</td>
<td>30</td>
</tr>
<tr>
<td>4 Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>4.1 Integration with the WERTi System</td>
<td>36</td>
</tr>
</tbody>
</table>
4.2 Method and Settings ............................................. 38
4.3 Output Examples ................................................. 40
4.4 Results and Discussion .......................................... 44
4.5 Performance ...................................................... 48

5 Future Work ....................................................... 49

6 Summary ............................................................ 52

A Bibliography ........................................................ 53

B List of Acronyms ................................................... 59

List of Figures

1 Syntactic representation of semantic–capturing patterns .......... 12
2 A schematic overview of the the ARCQC system’s architecture .... 14
3 TF-IDF score distribution calculated by ARCQC ..................... 22
4 A WERTi web-page with an additional button at the top for loading the quiz 37
5 An HTML/JavaScript quiz page generated by ARCQC ............. 38
6 Evaluation data summary plot ....................................... 46
7 Distribution of the generated tasks for the entire data ............. 47

List of Tables

1 All tags supported by the ARCQC system ............................ 33
2 Patterns with example sentences and questions ..................... 40
3 Evaluation data summary ............................................ 45
4 Classification of error types in tasks and distractors generation ... 45
1 Introduction

With a steeply growing selection of freely available authentic texts on the web and computation power greater than ever available on-demand, it is only natural that the field of Computational Linguistics (CL) is becoming more and more important and being integrated in an increasing number of areas of modern life.

CL — and its more applied side, Natural Language Processing (NLP) — deal with the interaction between natural, human languages and computers. Since computers play such a major role in our lives nowadays, it is required that they will be able to understand and interact with our language(s). It turns out that not only can they use them for interacting with us, but they can also help us learning them.

One of the relatively recent fields to make use of CL is education, particularly language learning (see e.g. Meurers (2012)). Intelligent Computer-Assisted Language Learning (ICALL) systems make use of linguistics knowledge, applications of language acquisition theories, educational and cognitive methods and other aspects necessary for successful learning. Such ICALL systems are slowly being put into use in schools and online language learning courses and assist both the teachers and the learners in the process of language acquisition (Dreyer and Nel, 2003).

Reading Comprehension (RC) is one of the aspects of language learning assessment. For RC activities, both a (usually unseen) text and tasks about the text are required. For the sake of terms clarification, in this context, a task is defined as “a classroom event that has coherence and unity, with a clear beginning and an end in which learners take an active role” (Cameron, 1997, p. 346).

All kinds of questions can be used for such an assessment. Two main groups of such questions are Recall Questions (RQs), which concentrate on recalling relatively shallow information about the read text, and inference question, where the learner needs to demonstrate deeper understanding of the material, make deductions etc. This paper deals merely
with RQs and any type of inference questions is out of its scope.

RQs come in different flavours. They can rely on general knowledge, knowledge acquired from previous read texts (i.e. long-term memory exercises), combination of several pieces of information and more. The RQs presented and discussed in this paper are for a text the learner has recently read, and therefore are designed to practice short-term retrieval, perform reading-check, and ideally verify acquisition of a basic knowledge about the main concepts of the text.

In this paper, an experimental approach for automated RQ generation is introduced, in which the way tasks are generated by the system can be customized by the user and are based merely on the given text itself. Such an approach should not only save teachers precious time spent on writing the tasks from scratch, but also give them some confidence that the students have actually read the text given to them and got the gist of it.

1.1 Motivation

It is very important for teachers to keep track of their students’ understanding of the material taught in a class. For a teacher it is nearly impossible to know at each given moment the level of understanding of each of the students in class. In the book *The Art and Science of Teaching* (Marzano, 2007), it is elaborately explained how teachers can engage their students and improve their concentration in class by, for example, letting them read in advance about the topic they are going to deal with in class.

While this might help, the teacher must somehow verify that the students have indeed read the text and acquired at least a basic understanding of it.

Such a verification could of course be achieved by giving the students various type of tasks (like closed-questions, descriptive tasks, writing tasks etc.) about the text, which require the goal level of understanding the teacher wishes to achieve. This process is problematic not only because it would require the teacher to go through all, or at least most, of
the students' works in order to get an idea about their understanding, but also due to the
tremendous amount of time the teacher would need for creating — and coming
up with answers for — all those tasks.

Seeing that, as a rule, the topic will be taught in detail in class anyway, the demon-
stration of a shallow understanding of it should suffice as a prerequisite for the class. This
way, the teacher knows the students are ready to learn about the topic with some basic
background already in mind.

Achieving such proven basic background and verifying that the students have read the
text is exactly where an Intelligent Tutoring System (ITS) may assist teachers. Automating
the quantitative task of generating recall and factual questions (including a way to check
the students’ answers to them) will shorten the time needed for such verification and
will leave more time for qualitative tasks to take place in class. Such a system would
assist teachers to support slower students in class, who need more time for
understanding texts (which they cannot get in class) and obviously save a lot
of time for the teachers, which they could then invest in tasks better carried out by
them than by computers.

1.2 Outline

This paper begins with a survey of some theoretical background about Reading Compre-
hension and Recall Questions, where the aspects of reading comprehension, and specifically
the use of RCs for testing it, is discussed. The part about Automated Question Genera-
tion presents some of the distinguished works in the field. Subsequently, The System —
ARCQC - developed concomitantly to this paper is introduced, along with its Task Gen-
eration, Overgeneration and Ranking and Customization and Generalization components.
Then, the Evaluation of the system is shown, as well as some interesting Output Examples.
Finally, several ideas for Future Work are suggested and a Summary of the entire paper
takes place.
2 Background and Related Works

Some work has been done in the field of Automatic Question Generation (AQG) for RC, both research and applied. On the research facet, different kind of tests and experiments are carried out, in order to determine the best way to test understanding of a text and to measure language level, while the applied facet introduces tools that make use of the scientific methods for practically help learners improving their RC competence. A survey of some of the important AQG systems is introduced in this section, as well as theoretical background and real-world examples of the use of RC and RQs.

2.1 Reading Comprehension and Recall Questions

The set of abilities that includes understanding of a text, making inferences from it, recall information from it and apply prior knowledge to it, and some other linguistic capabilities is the core of one’s reading comprehension level (cf. e.g. Rayner et al. (2001)). RC is one of the four language-use abilities taught is schools and are examined in different language tests: reading comprehension, listening comprehension, verbal expression and written expression. It is said that developing strong RC capabilities at an early age strongly affects the ability to deal with high-level texts later in life and is considered to be one of the most important skills that university students of English need to acquire (Levine et al., 2000).

Assessing RC level is not a simple task, and it becomes even more complex in the case of an heterogeneous group of learners, due to a variety of different factors, such as vocabulary size, prior knowledge, reading strategies and more, which affect RC level (Pearson and Gallagher, 1983). The 2013 worldwide test takers report\(^1\) of the two “passive” modules, listening and reading (in contrast to the “active” modules, speaking and writing, in which the learner needs to demonstrate productivity) of the Test of English for International

Communication (TOEIC) test\(^2\) reveals that the average score of the reading module is lower by about 12% from the average score of the listening module. This is somewhat surprising, seeing that listening encompasses one more aspect, namely time. Interestingly, the average scores of these modules on the Test Of English as a Foreign Language (TOEFL) test\(^3\) report\(^4\) for year 2010 show only a minor difference of 2%. This could be due to various factors and might point to the fact that people aiming for academic-level English have more experience in reading and possess more of the necessary skills for RC mentioned above.

In her well-known and incredibly detailed study, Durkin (1978) states that RC can be taught in class by giving relevant comprehensive instructions that will bring the learner to approach the text more effectively. Her study observed that teachers invest about 36% of their RC teaching time in activities, and 12% of that in comprehension activities, i.e. actual practice in RC. This considerable amount of time could be channeled towards activities that require more advanced teaching methods, and therefore better be performed by human teachers. This concern, along with some other practical ones, is (at least to some extent) dealt with using so-called “technology-enhanced learning environments”. An example for such an environment is presented by Dreyer and Nel (2003). For more information and examples of such environments, see Automated Question Generation.

RQs ( interchangeably referred to as factual questions) are a certain type of questions dealing with recalling information from memory and are typically “about reading materials, either from teachers or students, [and] are usually short and targeted at a single piece of information” (Heilman and Smith, 2010a). RQs are often used in RC tests like TOEIC

---

\(^2\)The Test of English for International Communication (TOEIC) is ”an English language test designed specifically to measure the everyday English skills of people working in an international environment.”

\(^3\)Test Of English as a Foreign Language (TOEFL) is a standardized test required from students who wish to study at American colleges and universities and whose native language is not English

\(^4\)https://www.ets.org/Media/Research/pdf/TOEFL-SUM-2010.pdf
and TOEFL. Unlike in the traditional sense of RQs where the information to be retrieved is stored in the learners’ long-term memory, in the context of RC they are used for quickly checking whether the learner is capable of retrieving the relevant piece of information from the correct place of a given text. This task is rather straight-forward and doesn’t require the learner to further analyze it or make deep inferences out of it.

Such questions come to answer two basic needs of the overall RC assessment: 1) to verify that the student has read the text at all and grasped at least a general understanding of it, and 2) that the student knows where to look for basic pieces of information provided in the text. This general orientation to the text familiarizes the learner with the essentials of it and by that prepares him/her better for class and lets him/her concentrate on higher level tasks that require a level of understanding beyond information recall.

This type of more technical task, which simply requires a lot of training — without continuous supervision by a human teacher — can be carried out by computer-aided ITSs, as demonstrated in Automated Question Generation.

2.2 Automated Question Generation

NLP methods are implemented into ITSs for creating newer, more sophisticated systems to help learners and teachers. ITSs have been developed for assisting learning in different disciplines, like physics (Graesser et al., 2004), grammar training (Chen et al., 2006), academic writing support (Liu et al., 2012) and more.

A good overview of the history, use and potential of ICALL systems is given by Warschauer and Healey (1998) and Yazdani (1986). Furthermore, Heilman (2011) gives a very comprehensive survey of the state of the art methods in AQG and its challenges.

One of the most distinguished works in the field of AQG was accomplished by Mitkov et al. (2006) (see also Mitkov and Ha (2003); Mitkov et al. (2009)), where a rule-based approach for AQG is realized. The system introduced in that paper is designed to deal with texts about a specific subject, namely linguistics, and generates Multiple Choice Question
(MCQ) based on their contents. In the term extraction phase, the Part of Speech (POS)-based regular expressions \([AN]+\) and \([AN]*NP[AN]*\) are used for detecting so-called key-terms consisting of nouns and noun-headed constituents. That means that any token matching either of these patterns will be considered a target, regardless of its context or syntactic role in the sentence.

As for distractor\(^5\) selection, the lexical database for English WordNet\(^6\) (Miller, 1995) is used for querying for semantically related terms. Terms from the text are taken into consideration as distractors only in the rare case that not enough of them have been found by WordNet. The system then generates questions of various forms (e.g. “which-kind-of” and “[predicate-]object”). The quality of the items generated by that system are reasonably good and reliable.

Other examples for works where semantics plays a major role in distractor selection are those by Papasalouros et al. (2008) and by Al-Yahya (2014). These works use the semantic web standard OWL\(^7\) for finding distractors based on ontology relations.

Another, rather simple, way to select distractors is to collect tokens with the same POS, as done by Coniam (2013).

A machine learning approach for AQG is introduced in the work done by Hoshino and Nakagawa (2005). In their work, the system learns from a corpus where to put a gap for generating Fill in the Blank (FIB) tasks with distractors (i.e. a combination of MCQs and FIB tasks). The tasks are based on BBC\(^8\) news articles, 200 – 500 words in length. Unlike in the training corpus where the distractors are, for example, other forms of a verb (the

\(^5\) Sometimes also called “distracter” in the literature of classical test theory. One of the incorrect answers presented as a choice in a multiple-choice test (www.thefreedictionary.com)

\(^6\) https://wordnet.princeton.edu/

\(^7\) http://www.w3.org/2001/sw/wiki/OWL

\(^8\) http://news.bbc.co.uk/
correct answer), the system randomly picks tokens from the text as distractors, without taking any syntactic or semantic constrains into consideration. This method uses the text itself as the source of distractors on the one hand, but on the other hand, due to the random distractor selection mechanism, it is likely to generate at least some distractors that can be easily excluded from being the correct answer.

Two very interesting systems that follow approaches different from all those mentioned so far are those presented by Agarwal et al. (2011) and Feeney and Heilman (2008). The latter tackles the issue of verifying that the learner has read the text by using vocabulary tasks of the type “choose the set of words related to the text”. In these semantic fields based tasks, the learner needs to select the set with the words most related to the content of the text, and by that show that he/she understand what the text is about.
3 The System — ARCQC

PROJECT Automatic Re-Call Questions Creator (ARCQC) is, in its essence, a somewhat more flexible and independent implementation of AQG methods introduced in the section Automated Question Generation. These flexibility and independence are achieved by two main corresponding key principles that are realized in the system:

- **Customization** — The tasks generated by ARCQC are defined by editable eXtensible Markup Language (XML)-based schemas (or *patterns*), as opposed to hard-coded, pre-defined regular expressions or token-based detection approaches. This principle is derived from the idea that the user (presumably a teacher or a person with educational knowledge) best knows the texts the system generates the tasks for. Therefore, instead of letting the system generate fixed forms of tasks, the user’s knowledge is used for helping the system generate more suitable tasks for the specific text, making them more relevant for the learners solving them.

  For instance, a biographical text is more likely to contain time spans and actions (performed by the figure) rather than names of corporations or definitions of terms, which are more likely to occur in an article of an economics newspaper. Based on this previous knowledge of the user about the contents of the text, the desired tasks will most probably differ. Customizing these patterns makes the system change the focus of the generated task and avoid (as much as possible) generation of irrelevant or off-topic tasks.

Moreover, the distractors selected for the MCQs are also determined by the pattern. This serves the goal, introduced in the Motivation, of confirming that the learner has read the text and got at least a basic factual knowledge about it. Selecting other types of distractors, like orthographically similar distractors, would not serve this purpose.

For more details on how the customization principle is realized and a full list of supported tags see Customization and Generalization.
• **Self-Contained Generation** — As explained later under Distractor Selection, AQG systems, just like other NLP-assisted programs, tend to rely on external resources when it comes to content. Such resources could be lexical dictionaries, semantic relation mappings and others.

Relying on such external resources for generating distractors for MCQs, might distances the task from the content of the text it is based on. Not only might this misrepresent the topic of the text to the learner, but it also makes the tasks easier to solve, seeing that such out-sourced distractors are likely not to occur in the text at all (however semantically related), which makes it very easy to directly rule them out as the correct answer, even if the learner has only shallowly read the text or simply has a superficial knowledge on the topic.

In ARCQC, the whole generation process (from target detection to distractor selection) is based merely on the given text (except for very specific cases as explained in Term Extraction and in Distractor Selection). That means, for instance, that targets will be determined according to their relevancy to the specific text and that distractors will be taken from within the text (based on their semantic, syntactic, or functional role). In order to preserve this principle, even in rare cases in which the text does not contain enough relevant distractors, they will be taken from previously used text (assuming that text that were used in sequence are more likely to be topic-related) and not from fixed, external resources.

One of the main challenges of every AQG system is to capture the semantic content of a text using mostly (if not only) structure-based analysis, as very well demonstrated in the works of Curto et al. (2011), Curto et al. (2012), and Mazidi and Nielsen (2014) — all the more so when dealing with authentic, real-world texts. Such texts can deal with any topic, use any vocabulary, syntactic and readability levels, and might even contain non-standard, ill-formed language structures. Therefore, it goes without saying that the user must first be aware of the capabilities and limitations of the system in question. Secondly, it’s an
open secret that capturing meaning is still a difficult task for NLP programs, especially when it comes to arbitrary authentic texts.

To overcome this problem, the XML-based patterns used in ARCQC function as an interface, through which the user — for whom understanding semantics and context is trivial — can create this link between structure and meaning. This way, both the human being and the computer contribute by doing what they do best. The human being understands the meaning of language and the computer analyzes and processes text accurately and fast.

For example, for capturing the meaning of a definition or an is-a relation, the user may create a pattern that identifies a noun phrase, followed by a “meaning-assigning” verb (like “is”, “becomes”, “means” etc.), followed by another noun phrase or perhaps just a free text. Such a customized pattern links between the structure of a sentence in the text and the meaning of a word in it, while the system is not required to grasp this meaning using deep semantic analysis or external lexical dictionaries. This pattern (and variations of it) is proved to capture this kind of semantics, since it is also used for extracting term lattices for ontology databases (Navigli and Velardi, 2010, p. 1320-1321). A similar pattern is shown by (Kalady et al., 2010, p. 8), but the resulting questions of that work might be ambiguous. One of the examples presented there is the question “What is a volcano?”, which could refer to a general, free definition of volcano or to a specific definition presented in the text. These two answers are likely to be similar, but still differ, and without a question answering algorithm this similarity might be missed. This issue is solved in ARCQC by letting the user add guidance text to the generated question using specific tags in the pattern (see Pattern Structure, like “In the text, […]” or “According to the definition found in the text, […]”).

Another pattern could, for instance, capture the the semantic construction of a person who performed some an action. While such construction has a meaning that the learner needs to grasp, the underlying structure represented in the corresponding pattern may consist of a noun phrase detected as a proper name (i.e. a person-type Named Entity (NE),
Figure 1: Syntactic trees demonstrating structures captured by Subject-Predicate and Definition/Is-A patterns. In red are the targets. MCQs that can be generated for these targets are “Who drank an espresso on the table?” and “What is a Wug?” followed by an active verb (in order not to include the passive form, which will lose the targeted meaning. Verbs like “be” and “have” may be excluded for assuring the capture of an actual action), potentially followed by a noun phrase (for capturing transitive verbs) or an adverbial phrase (for capturing manner). This pattern may of course get more involved in order to narrow down the targeted meaning. More complex patterns may capture more sophisticated structures, representing more specific contents, like time spans, tenses, quantities and others.

3.1 Architecture Overview

ARCQC includes all the traditional modules of an AQG system, such as Text Analysis, Term Extraction, Target Detection, Instruction Formulation and Distractor Selection as well as some additional modules for supporting its more unique features.

As can be seen in figure 2, the first module in the Quiz Generation component is the
Schema Processor. This module is responsible for extracting all the necessary information from the XML-based schema (see Customization and Generalization) and transferring it to all the other modules in this component. After all the tasks were generated, they are being filtered and then ranked by the ranking module (see Overgeneration and Ranking). This process results in a quiz with a pre-defined (configurable) number of tasks.

The whole task generation process is initiated and supervised by the Manager module, which is the start and end point of the entire system.

All of the external resources are managed by the Resources component, which is responsible for reading, writing, loading and processing them. All of these resources are external files that must exist in order for the system to work (some of them are automatically created in their default form in case of absence). In addition to these external resources, a log file is also managed by the Manager module. This log file traces all the activities done by the different system’s modules, including all errors and problematic outputs generated by them. The threshold of severity level for these log entries is also configurable.

The architecture presented in figure 2 includes additional modules that were added to ARCQC in order to let it interact with the Working with English Real Text (interactive) (WERTi) (Meurers et al. 2010; Metcalf and Meurers 2006) system, which was used for Evaluation (see Integration with the WERTi System).

3.2 Text Analysis

Like any NLP tool that deals with free text, ARCQC first needs to analyze the text given to it. Some NLP tasks, like AQG, require more extensive analysis, and sometimes even need to use external resource in order to achieve the desired level of performance.

The text analysis is the first process to be carried out on the way to generating a set of questions for a text. Various tools are used throughout this process, but the main
ones are Stanford CoreNLP\(^9\) (Manning et al. (2014)) and Apache OpenNLP\(^{10}\). The reason for using two different comprehensive libraries are design-driven and technical in nature. For instance, the different modules they contain, flexibility of their models, their APIs etc. Although these two libraries are not originally designed to interact with each other, the system does use both of them interchangeably (for different purposes) and pipes the outputs of the modules together.

---

\(^9\)http://nlp.stanford.edu/software/corenlp.shtml
\(^{10}\)https://opennlp.apache.org/
First, the text is segmented into sentences, since the XML patterns (see Customization and Generalization) are always at the sentence level. These sentences are also the linguistic units that will be compared to those patterns, and therefore treated as the highest level of analysis (with the exception of co-referencing). Contents inside any kind of parentheses are removed, since that as a rule these usually contain only additional information and are not the most important parts of the text. Special symbols like asterisks, underscores and slashes are removed as well, for they do not add any substantial content to the text. Seeing that appositions are not removed from the text, this step provides some very basic simplification and clean-up of the text. Subsequently, the sentences are tokenized and POS-tagged. Then, some other analyses might be performed if required by the pattern. Finally, some more advanced analysis processes like co-reference mapping and name entity recognition are performed.

For instance, co-references are used for enlarging the set of sentences that could be targets for tasks and for making the detected terms and selected distractors more reasonable. The use of co-references also partially overcome the need of sentence splitting, e.g. as done in the work of Kalady et al. (2010) and Heilman and Smith (2010a). Name entities are used for auto-detecting term (name entities are automatically counted as important terms, as described in Target Detection), for selecting better distractors (for example not giving location-entities as distractors of a person-entity term), and of course as a possible constraint for an XML task-pattern (like questions that are to be asked only on time-entities or modifying the instruction of the task based on the entity type of the term, like using the question word “who” instead of “what”).

Dynamic Analysis
Seeing that, as explained above, only some general, common analyses are required as a prerequisite for starting the tasks generation process, while some more detailed and sentence-specific analyses are not required for all tasks, and in order to prevent unnecessary heavy processing to take place, some of these more specific analyses may only be performed
on-demand (typically based on a pattern) and on-the-fly, i.e. while generating a task. This is as opposed to in the pre-processing of the whole text (see figure 2). Such dynamic analysis can be used, for example, in the case of chunking. Not all sentences are needed to be chunked in order to start creating tasks. However, chunking might be required by a constraint in an XML pattern, e.g. a pattern that extracts the task’s term by referring to its chunk type or position in the sentence. Another example is Term Frequency — Inverted Document Frequency (TF-IDF) (see Term Extraction) scores, which are required to be calculated only for extracted terms (and even then, not always) and therefore will be calculated only when requested — in oppose to calculating it for all the tokens in the text anyway, which would have cost unnecessary processing.

3.3 Task Generation

Task generation is the core of any AQG system. In ARQCQ, this process is performed after the initial text analysis, in a loop, throughout which the entire text is scanned sequentially. Each sentence is scanned based on each of the patterns for detecting potential targets. Subsequently, the main term is extracted from the target sentence based on the pattern and the target sentence is transformed into a task. Finally, additional processes are performed (like distractor selection), if necessary, and the task is added to the quiz under creation. Some additional filtering and verifications are performed after the whole process is completed, in order to make sure that all of the generated tasks are valid, complete and solvable.

3.3.1 Target Detection

The first step in generating tasks from a given text is to determine which parts of the text are suitable for this purpose. Subsequently, targets (in this context sentences) matching this definition are detected. Finally, tasks for these targets are created. The algorithm works as follows:
Algorithm 1 Target Detection

1: procedure Find suitable targets (Text T, XMLPatterns M)
2:   for sentence s ∈ T do
3:     for pattern p ∈ M do
4:       if s matches p then
5:         Generate task for s based on p
6:     end if
7:   end for
8: end procedure

In ARCQC, the patterns that define potential targets are XML-based and are stored in an external, editable file, which grants a high degree of flexibility to the user to choose what kind of targets shall be considered for generating tasks. For more details about this customization see Customization and Generalization.

3.3.2 Term Extraction

After a target sentence is detected, the term (i.e. the correct answer for the to-be-generated task) is extracted. Each sentence that has been detected as matching a pattern (see Target Detection) is further analyzed based on this pattern, and then the term is extracted according to the <Target> tag of the pattern (see Customization and Generalization). There, a term can be defined as one or more words, a chunk (phrase) of a specific type, a specific NE type and other criteria. The pattern-approved potential terms are then filtered, in order to give focus to the most statistically significant (and therefore probably also the most important) terms in the text. To do so, the TF-IDF\textsuperscript{11} (Rajaraman et al., 2012, Chapter “Data Mining” p. 7-9) scores of the potential terms are taken into account. In ARCQC, the calculation of TF-IDF scores is stretched over three phases:

\textsuperscript{11}a numerical statistic, often used in Information Retrieval and Text Mining, that reflects how important a term is to a document in a collection of documents
1. Corpus-Based Static IDF Score

The salience of a word in a specific text can only be determined when compared to its general salience in the language, which is determined by its frequency in the language. For this, a corpus is needed for representing the language, and the inverted document frequency (IDF) is calculated using the formula

\[
IDF(t, D) = \log \left( \frac{N}{1 + \left( \sum_{n=1}^{N} \left[ t \in d_n \right] \right)} \right)
\]  

(3.1)

where \( D \) is the corpus, \( N \) is the number of documents in the corpus, \( d \) is a document in the corpus and \( t \) is the term in question. The result of this calculation is the logarithmic of the number of documents in which \( t \) occurs at least once, i.e. the term’s IDF score. Subsequently, the resulted score is filtered by a minimal-score threshold, in order to deal with a smaller scale set of terms.

\[
f(x, \text{lim}) = \begin{cases} 
\text{filter}, & \text{if } x < \text{lim} \\
\text{retain}, & \text{otherwise.} 
\end{cases}
\]  

(3.2)

In the final version, this threshold was arbitrarily set to 13, which resulted in filtering out 89.54% of the terms due to their negligible scores. Still, the number of remained terms was more than enough for the purpose of the system.

The corpus that was used for creating these scores is PukWaC (Baroni et al. (2009); Johansson (2008)), which is based on the ukWaC\(^{12}\) corpus. This corpus contains 2,692,642 documents containing 12,589,018 different terms (i.e. types). Due to this high number, the terms were extracted as tokens and not as lemmas, making the

\(^{12}\)A 2-billion word corpus constructed from the Web limiting the crawl to the .uk domain and using medium-frequency words from the British National Corpus (BNC) as seeds (see http://wacky.sslmit.unibo.it/doku.php?id=corpora for more details)
scores more realistic and independent from tokens with similar lemma. This technique is similar to the one used by Feeney and Heilman (2008), but with slightly different formulae and a much larger corpus.

2. **Text-Based Dynamic TF Score**

In the second phase, the TF score is calculated, based on the text for which tasks are to be generated. Seeing that this text is not one of the documents in the corpus (see above), it is expected that not all the term in it will appear in the corpus. As a result, not all terms with a TF score will also have an IDF score, which is of course necessary for calculating the final TF-IDF score. This problem was solved by adding a highly rewarding factor to the IDF scores of terms that occur in the text, but do not occur in the corpus, assuming that such terms are indeed specific to the text and therefore are probably more significant. The formula used for calculating the TF score of terms is

\[
TF(t, d) = \frac{\sum_{n=1}^{W} [w_n \in d = t]}{W} + B \cdot IDF(t, D) = \log N
\]

(3.3)

where \(d\) is the text to generate the question about, \(W\) is the number of tokens in \(d\), \(w\) is a token in \(d\), \(t\) is the term is question and \(B\) is the “bonus” for being a unique term. In this calculation as well, the tokens are taken as they appear in the text, i.e. case sensitive, not lemmatized etc.

3. **TF-IDF Calculation**

After both the IDF and TF scores are calculated, the final TF-IDF score can be calculated. This is done using the formula

\[
TFIDF = (1 + TF(\text{term})) \cdot IDF(\text{term})
\]

(3.4)
The addition of 1 to the TF score guarantees that scores of terms that do not occur in the text will not be equal to 0 (considering that the IDF score is smoothed as well, as described above), which will result in a score based on the IDF score alone. While in traditional usage of TF-IDF there is no meaning to scores of terms that do not occur in the corpus, in ARCQC such score might be used when generating distractors and ranking tasks (see Overgeneration and Ranking), and therefore the +1 is necessary.

The terms with the highest scores are considered to be the significant terms of the text. The scores are then filtered by a threshold, which is the average of the TF-IDF scores of all the the tokens in the text. Name entities are automatically considered as important terms, regardless of their ranking. Very often the score of a multiple-words term is required (e.g. a phrase, NE and so on). In this case, the average of the scores of all of the parts (tokens) of the multiple-words term is the score of the whole term.

The whole term extraction process is summarized in the Term Extraction algorithm.

\[\text{Algorithm 2 Term Extraction}\]

\begin{verbatim}
1: procedure EXTRACT CORRECT ANSWER(Pattern P, Sentence S, Text T)
2:     answer ← extractTermTag(P)
3:     if not answer ∈ NE_T then
4:         answerCoref ← getCoref(answer) \(\triangleright\) if no coref, assign term itself
5:         answer ← termOfMax(TFIDF\text{answer}, TFIDF\text{answerCoref})
6:     end if
7:     if not TFIDF\text{answer} is significant then
8:         return NULL \(\triangleright\) don’t create task for this term
9:     end if
10:    return answer \(\triangleright\) use answer as term of task
11: end procedure
\end{verbatim}
For the sake of demonstrating the feasibility of this TF-IDF scoring method, the ten terms with the highest score (on the left) and the ten terms with the lowest score (to the right) of the nouns in the text extracted from the Wikipedia entry about Reading Comprehension\(^{13}\) are listed below:

<table>
<thead>
<tr>
<th>Manzo</th>
<th>self-correcting</th>
<th>people</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence-level</td>
<td>Craik</td>
<td>work</td>
<td>development</td>
</tr>
<tr>
<td>text</td>
<td>pragmatics</td>
<td>years</td>
<td>level</td>
</tr>
<tr>
<td>read-test</td>
<td>Comprehension</td>
<td>number</td>
<td>process</td>
</tr>
<tr>
<td>narrative-level</td>
<td>Questioning</td>
<td>area</td>
<td>research</td>
</tr>
</tbody>
</table>

The terms *Manzo* and *Craik* are names of researchers in the field, making them relevant to the text. The word *reading* has a relatively low score, for it is by itself not that significant. However, the collocation *reading comprehension* will get a much higher score (equal to more or less the 15\(^{th}\) – 20\(^{th}\) place), seeing that ARCQC takes all parts of a multiple-words term into account when calculating its TF-IDF score. This ensures that this collocation, which is obviously very relevant in this text, will be a good candidate as a target in the quiz generation process.

Lower scores reflect less significance or uniqueness of the term, and indeed the terms with the lowest scores on the list are generally not directly related or unique for the topic (at least not by themselves).

The distribution of the TF-IDF scores of terms with all POSs is shown in Figure 3.

As a final remark, it is worth mentioning that in addition to increasing the chance of using the most relevant terms from the text for creating a quiz, the use of significance-based filtering automatically prevents functional words and reference words (like “this”, “him”,

---

\(^{13}\)https://en.wikipedia.org/wiki/Reading_comprehension
“I” etc.) from being used as a term for a task, without the need to explicitly defined a set of these words.

### 3.3.3 Instruction Formulation

In ARCQC, tasks are generated on the sentence level and sentences are detected as targets based on patterns. In those XML patterns there is a special tag, `<Produce>`, for defining the way the sentence will be transformed into a task (see Pattern Structure).

Seeing that ARCQC is designed to generate RQ, the transformation is usually fairly straight-forward and therefore can be defined by a few steps. First, all of the desired target (i.e. parts of the original sentence) are detected in the Target Detection phase. Each defined target can then be manipulated in order to formulate the instruction of the task. A target can be replaced, follow a fixed text or be followed by a fixed text (multiple manipulations possible). Furthermore, each of these manipulations may or may not need to meet a condition in order to be executed.

For example, “definition” pattern discussed in Term Extraction may capture noun-phrases with animate or non-animated subjects. The sentences
(1) *Professor Quantum is cool*

(2) *balloons are cool*

will both be captured. However, while the former will require the word “Who” to substitute for the subject, the latter will require the word “What”. Seeing that both ways will be captured in the same way (as noun-phrases), this is solved by conditioning the first case with the noun-phrase being a NE of type person and leaving the second case as default in case the condition is not met. Subsequently, the pattern may define addition of text before or after any of the targets. For instance, the text “Out of these four possible answers,” may be added before the target, and the text “, according to the text,” after it. With more involved patterns, more sophisticated formulations can be made. The targets are then processed **sequentially, according to their defined indices.** Finally, the resulting instruction for sentence (1) would be *Out of these four possible answers, who, according to the text, is cool?*. 

Notice that the example above started with the whole original sentence as the starting point for the instruction formulation. The other possible starting point is nothing. In that case, the whole formulation needs to be defined in the pattern, using the captured targets and manipulations on them (remember that anything can be a target, and not only the correct answer, which is defined separately; see Pattern Structure for more details).

### 3.3.4 Distractor Selection

For **MCQs**, \(n - 1\) of distractors are required (where \(n\) is the number of choices offered to the learner as possible answers). The role of distractors is to verify the precision of the learner’s knowledge. Therefore, they ought to challenge the learner, while leaving only one of the choices the (most) correct one. This is typically done by making each distractor similar to the answer in at least one aspect (see examples in Automated Question Generation. Such aspects include, among others, orthography, meaning, semantic role, tense (for verbs) and
context. It goes without saying that the latter is text-specific by nature, which requires knowledge about the specific given text (in oppose to general knowledge or knowledge based on a series of texts) and therefore best suits to the purpose of ARCQC.

As explained before, one of the main principles of ARCQC is to base the generated tasks (including their distractors) merely on the text in question, and thus the context aspect is very relevant. Seeing that ARCQC aims to present RQs to the learner, any task whose answer is not in the text already borders the domain of inference questions, which are out of the system’s scope. Furthermore, taking distractors from the text prevents the examinee from excluding a choice simply because it doesn’t appear in the text at all (as can be done, for example, in the work of Mitkov and Ha (2003)), and thus the distractors are strongly preferred to come from within the text itself.

The selection of appropriate distractors is the last sub-process of generating a task (see figure 2 on page 14), and it is generally done in two steps:

1. **Setting criteria** — Various information about the target term of the task (see Target Detection) is taken into consideration, like phrase type, NE type (if applicable) and other criteria that can be defined in the pattern based on which the task was generated. Generally, the system prefers more specific criteria over less specific. For instance, for the token *New York*, the location NE criteria will be favored over it being a noun phrase.

Seeing that the patterns used in ARCQC refer to form and that semantically similar words tend to occur in similar environments\(^{14}\), this will increases the chances that the distractors will be semantically close, making the task more challenging.

2. **Collecting distractors** — Based on the set criteria, the Text Analysis module (see figure 2 on page 14) is queried for tokens that match them (see Text Analysis). These

\(^{14}\)“You shall know a word by the company it keeps” (J. R. Firth, Selected Papers, 1957)
tokens are then filtered based on their TF-IDF score (see Term Extraction) in order not to use distractors that are not very relevant in the given text.

One further mechanism used in this phase is co-referencing. This is done in order not to lose good distractors that are not explicitly mentioned. For example, from the pair of sequential sentences

(3) New York is in the USA

(4) It is very big

it is trivial to generate the question What (city) is in the USA?. However, the straight-forward analysis would take the token it from the second sentence (assuming it matches the criteria), which will most probably be filtered out by the TF-IDF filter. With the co-referencing mechanism, this token will be detected as referring to the NE New York and the latter, having a much greater chance to be significant in the text, will be extracted as a term, which might also be used for distractor selection and as a result broaden the variety of terms and distractors that can be extracted from the text and can be used for generating tasks.

Using only tokens from the given text as terms and distractors might also be the weak spot of the system. Shorter text are likely to contain fewer tokens matching each of the criteria. This makes it harder to find enough distractors for a task and especially harder to generate tasks with different distractors. For solving this issue, compromises regarding the self-contained generation must be made, and in the rare case in which not enough distractors can be extracted from the text, the system will also query for matching tokens extracted from previously used texts (taking only the minimal amount required for presenting the full list of distractors required). These are automatically stored by the system, so that there can be used even when those texts are no longer available. The assumption is that those texts will often be, at least to some extent, related to the current text, because people are more likely to search
for texts about the same topic, from the same resource etc. Thus, this compromise is expected to better serve the purpose of the system, seeing that it indeed takes tokens not from the given text on the one hand, but still does not rely on external, totally unrelated sources, on the other.

Finally, in the very unlikely case in which after everything there are still fewer distractors for the task than required (configurable in the configuration file), the task will simply present fewer choices to the learner.

3.4 Overgeneration and Ranking

Computers and humans process data differently and have different advantages. Particularly, they use different kind of approaches and techniques. While humans beings are better, for example, at quickly ruling out tasks that do not reflect the required knowledge and choose more sophisticated distractors, computers are better in processing large amount of data fast and storing a large amount of data in their memory. It could be generally said that human beings are good at qualitative tasks while computers perform better in quantitative tasks.

Due to these differences, an AQG system must take advantage of the better sides of computers. First and foremost, processing a large amounts of data precisely and fast. Secondly, storing large amounts of data and connecting to other resources is a simple task for modern computers.

In ARQC, these are taken advantage of by deeply analyzing the text, matching sentences and patterns, using other resources like dynamic external lists for getting additional information to use in the generation process and more. Ultimately, all of the above results in a system that can generate a long list of relevant questions, but unfortunately cannot decide which of those are the best questions to eventually display to the learner. That is, more questions are generated than actually needed, i.e. overgeneration. As described in Heilman (2011), such overgeneration may be filtered by a ranking mechanism.
Smith (2010) presents a “categorical” ranking approach where tasks are ruled out if they fall under any deficiency criteria like ungrammaticality, missing answer, vagueness and more. This approach concentrates on ruling out questions rather than ranking those that are valid. Another, more linguistic-oriented approach is used in the work of Becker et al. (2012), where semantic and syntactic features are taken into account, rather than techniques used in information retrieval.

The ranking mechanism in ARCQC differs from these approaches, for it concentrates on ranking the quality of the valid tasks (invalid tasks are being excluded before the ranking process starts) and by being mostly based on TF-IDF scores, which are used in information retrieval. This mechanism does not filter out tasks, but rather gives “better” tasks a greater chance of being chosen. In other words, even the task with the lowest ranking score may be probabilistically chosen. The only exclusion made is for tasks referring to very long sentences. The threshold length was set to 21 words, approximately 3 times the average length found in the Microsoft Research Question-Answering Corpus\textsuperscript{15}, where the average length of questions is 7.2 words (Heilman and Smith, 2010a). In order for the ranking process to be fair toward all of the tasks, it is carried out after all of the tasks have been generated, so that the ranking scale is set and no new extreme values can appear and distort it.

Each task’s raw ranking score is calculated according to the following formula:

\[ \text{Rank}(\text{task}) = \log \left( \frac{1}{c} \sum_{n=1}^{c} TFIDF(choice_n) \cdot \text{length} \right) \]  \hspace{1cm} (3.5)

where \( c \) is the number of choices the task presents to the user and \( \text{length} \) is the length, in words, of the instructions for the task. That means that tasks that use more relevant terms and have shorter instructions are preferred, in order to make the tasks easier to read.

\textsuperscript{15}Available at http://research.microsoft.com
As mentioned, the ranking score lets all of the tasks be chosen (rather than being out-filtered). For this, a scale needs to be used for determining each task’s probability of being selected. By default, the scale of 0-100 is used (the values themselves are meaningless; this scale simply makes it easier to think of the scale as probabilities). This is done by using the re-scaling formula:

\[ y = \left( \frac{x + \text{min}_{\text{negative}}}{\text{max}_{\text{value}}} \right) \cdot \text{max}_{\text{scale}} \]  

(3.6)

where \( x \) is the raw ranking score, \( \text{min}_{\text{negative}} \) is the lowest raw negative ranking score (if there is none, this is set to 0), \( \text{max}_{\text{value}} \) is the highest raw ranking score, \( \text{max}_{\text{scale}} \) is the highest value of the target scale and \( y \) is the resulting re-scaled ranking score.

Then, when tasks are chosen from all of the tasks generated for a quiz, they will be probabilistically selected according to these re-scaled probabilities. The number of tasks that will be included in the quiz is defined in the configuration file. The probabilistic option can be turned off as well. In this case, all tasks will have an equal chance of being selected, instead of based on their ranking score.

The whole term extraction process is summarized in the Overgeneration and Ranking algorithm.

### 3.5 Customization and Generalization

Different topics and different texts have different characteristics that influence the variety of questions that can be generated from them. Furthermore, different teachers may want to focus and ask questions about different aspects and parts of the text.

It is usually possible to post-edit the output of AQG systems (see e.g. Mitkov et al. (2006)) in order to give the teacher the possibility to shape the generated quiz to match
Algorithm 3 Overgeneration and Ranking

```plaintext
procedure CHOOSE_TASKS(Quiz Q, TargetScale S)
    if probabilistic then
        for task \( t \in Q \) do
            calculate(RawRankingScore\(_t\))
        end for
        \( r \leftarrow \text{Calculate}(\text{RawScale}(q)) \)
        \( \text{Rescale}(r, S) \) \( \triangleright \) \( S \) is by default 0-100
        for task \( t \in Q \) do
            calculate(ScaledRankingScore\(_t\))
        end for
    end if
    \( n \leftarrow \text{Random}(S_{\text{min}}, S_{\text{max}}) \)
    \( t' \leftarrow \text{GetRandomTask}(Q) \)
    if \( \text{rank}_t \geq n \) then
        \( \text{Choose}(t') \)
    end if
end while
else
    \( t' \leftarrow \text{GetRandomTask}(Q) \)
    \( \text{Choose}(t') \)
end if
end procedure
```

his/her preferences. However, this might take time and in some ways it is not very different, at least to some extent, from simply writing it by hand to begin with. Such post-editing also makes the auto-generation less automatic and might not use the power an automated system offers. By that the teachers return to the starting point, where they needed to come up with questions and/or answers (in the case of MCQs, for example) themselves.

In ARQC, questions are generated based on customizable XML-based patterns. Naturally, it is also possible to add completely new patterns and let the system generate questions according to them, and more importantly also to store and reuse them (unlike post-editing, which is relevant for a specific quiz only). Since the output of the system is
an editable file (either plain-text or HTML/JavaScript), post-editing is still possible, but ideally the need for it is kept to a minimum.

3.5.1 Pattern Structure

As mentioned before, the tasks generated by ARCQC are based on patterns, which the user can modify according to the nature of the text and the desired style of the quiz. These patterns are the interface that links the user’s preferences and the final output of the system.

Each AQG system needs to have some kind of a mechanism to detect target sentences and another mechanism for transforming them into questions. For instance, Curto et al. (2011) shows very clearly how syntactic structures capture semantic meaning and can be used for extracting terms and generating questions. However, it uses only POS-based rules, which limits a bit the variety and richness of structures it can capture. Further more, the work presented by Mazidi and Nielsen (2014) resembles the approach presented in ARCQC by making use of linguistic patterns as the core mechanism the whole AQG process is based on.

Table 1 summarizes the tags these patterns may contain, along with their attributes and dependencies:
<table>
<thead>
<tr>
<th>Name</th>
<th>Parent</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaData</td>
<td>Patterns</td>
<td></td>
<td>This is a tag under which any other tag can come for providing information about the collection of patterns, like <code>&lt;Author&gt;</code>, <code>&lt;CreationDate&gt;</code> and so on.</td>
</tr>
<tr>
<td>Pattern</td>
<td>Patterns</td>
<td>name: free text; the name of the pattern. This is the internal name that will be used in the system (e.g. in the log file, etc.)</td>
<td>The head tag of the whole pattern.</td>
</tr>
<tr>
<td>Properties</td>
<td>Pattern</td>
<td></td>
<td>The head tag of the pattern’s properties</td>
</tr>
<tr>
<td>Description</td>
<td>Properties</td>
<td></td>
<td>A free-text description of the pattern. This description is merely for giving the user information about the nature of this pattern and is not used internally by the system.</td>
</tr>
<tr>
<td>Types</td>
<td>Properties</td>
<td></td>
<td>A comma-separated list of the task types to be generated for this pattern.</td>
</tr>
<tr>
<td>Schema</td>
<td>Pattern</td>
<td></td>
<td>The head tag of all of the to be taken into account when generating tasks for this pattern — both for the detection and for the formulation phases.</td>
</tr>
<tr>
<td>Detect</td>
<td>Schema</td>
<td></td>
<td>The head tag of conditions that shall be detected in a sentence in order for it to match the pattern.</td>
</tr>
<tr>
<td>Target</td>
<td>Detect</td>
<td>number: integer; the index of the target. indices may be non-sequential, but must be unique. The task will be formulated in increasing order of the indices</td>
<td>Marks whatever its child-node(s) capture(s) as a target, which can then be referred to as the correct answer or applying manipulation on it.</td>
</tr>
<tr>
<td>Entity</td>
<td>Detect</td>
<td>Target</td>
<td>type: [Time</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Chunk</td>
<td>Detect</td>
<td>Target</td>
<td>phrase: regex (supported phrases: NP, VP, PP, ADJP, ADVP, SBAR, CONJP); the phrase type of the chunk (or whole sentence) position: integer; the phrase index this phrase must be in. If no position is specified, the whole sentence is examined text: regex; a specific text this chunk should contain</td>
</tr>
<tr>
<td>Produce</td>
<td>Schema</td>
<td></td>
<td>base: [base</td>
</tr>
<tr>
<td>Answer</td>
<td>Produce</td>
<td></td>
<td>target: integer; the index of the target to be used as the correct answer for the task</td>
</tr>
<tr>
<td>Tag</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| Modification | Produce | action: [add-before|replace|add-after]; the type of modification to apply to the original sentence  
|           |        | target: integer; the index of the target to carry out the operation on  
|           |        | condition: any entity or phrase type; optional, a condition to be met in order for this operation to take place  
|           |        | represents a modification of the original sentence to be made for generating the task’s instruction  
| Opening   | Produce | The text to put at the beginning of the task’s instruction (regardless the targets). This can be used for defining what the learner should do in the task (e.g. “Choose the correct answer”). The sequence “\n” breaks and feed a line.  
| Closure   | Produce | The same as the Closure tag, but at the end of the instruction.  

Table 1: All tags supported by the ARCQC system
The pattern shown on the next page demonstrates the use of some of the features described in Table 1. As written in its description, it will capture sentences of the form NP is NP, where the second NP may be a complex NP with multiple NPs and PPs in it. For instance, this pattern will capture the sentence “New York is a state in the Northeastern and Mid-Atlantic regions of the United States”.

The tree below shows the different chunks that the system will detect for this sentence using that pattern. The targets’ indices are attached to the corresponding chunks and the chunk with the frame around it is the chunk defined as the correct answer.
<Pattern name="What is something is">
  <Properties>
    <Description>Captures sentences of the form "X is Y"</Description>
    <Types>MCQ</Types>
  </Properties>
  <Schema>
    <Detect>
      <Chunks>
        <Chunk phrase="NPVPN [NP|PP]*"/>
        <Chunk phrase="VP" position="2" text="is|are|was|were"/>
        <Target number="1">
          <Chunk phrase="NP" position="1"/>
        </Target>
      </Chunks>
    </Detect>
    <Produce base="all">
      <Answer target="1"/>What</Answer>
      <Modification action="replace" target="1">Who</Modification>
    </Produce>
    <Opening>(Choose the correct answer)
    </Opening>
    <Closure>?</Closure>
  </Schema>
</Pattern>
4 Evaluation

One of the greatest challenges for an AQG system (and probably for any NLP application) is dealing with free, real-world texts. To test how ARCQC faces this challenge, the WERTi system was used for collecting authentic short texts (see Integration with the WERTi System). These texts were subsequently processed by ARCQC and quizzes were generated for them. The number of tasks generated was counted and their quality was measured.

In Output Examples some noteworthy outputs are discussed, and in Results and Discussion the final statistics are shown.

4.1 Integration with the WERTi System

To evaluate the system, a set of authentic texts needs to be acquired. In order to provide a dynamic, objective way to acquire them, the independent system WERTi\(^\text{16}\) (Meurers et al. (2010); Metcalf and Meurers (2006)) was used.

WERTi is a web application that queries news articles from Reuters News\(^\text{17}\). These articles are then processed and turned into online language exercises, such as coloring, FIB and MCQs for different linguistic targets that can be selected by the user.

As mentioned in the Architecture Overview, additional functionalities needed to be added to ARCQC in order for it to interact with the WERTi system.

First, processed web-pages with an activity contain an additional button (see figure 4) for loading a page with tasks based on the text in the page. Once such a page is loaded onto the web browser, the text of the page is extracted from the HTML structure and exported. This triggers ARCQC to be called in the background (while the user can complete all the tasks on the page) and use the extracted text as the input text to generate a quiz based on it.

\(^{16}\)http://sifnos.sfs.uni-tuebingen.de/WERTi/

\(^{17}\)http://www.reuters.com/
Subsequently, ARCQC generates the quiz for the provided text and exports it to an HTML/JavaScript page (as opposed to a plain text format when ARCQC is used independently). This format of output allows the HTTP server running WERTi to directly load the quiz and interact with it (e.g. via interactive drop-down lists for MCQs and input text field for FIB tasks). Finally, once the quiz is ready, the user can navigate to it by clicking on the aforementioned button.

The feedback provided for the MCQs and FIB tasks generated by ARCQC for WERTi keeps the standard interactive feedback of WERTi: green for a correct answer and red for a wrong one. Additionally, a “hint” button is always placed right beside the instance of the task in the page. When it is clicked, the correct answer will appear in blue (see figure 5).
4.2 Method and Settings

Two criteria were chosen for evaluating the system: the number of tasks generated and the validness (appropriateness) of the generated tasks and their corresponding selected distractors. Validness refers both to grammaticality and to the relevancy of the task. The WERTi system was used for collecting five authentic texts from each of four different topics (Politics, Sports, Business and Technology). These texts are the top hits of the integrated Reuters News search of WERTi, so that no subjective judgment played a role in choosing the texts, nor were they in any way modified to be more easily analyzed by ARCQC. As it happens, one article came up in the top results of both the Technology and the Business categories.

To make it more interesting, additional ten texts from both English Wikipedia and Simple English Wikipedia\textsuperscript{18} were used as well. These texts have a different structure from news articles and therefore were expected to yield different results.

\textsuperscript{18}https://simple.wikipedia.org
Additionally, the number of tasks generated by each of the 7 patterns used for evaluation was counted, as well as the average word per text in each of the categories, the average number of tasks per text, and the percentage of valid tasks and distractors generated. Open FIB tasks were not used for evaluation and the number of choices per MCQ was set to 4. The number of tasks per quiz was not limited, so that all generated tasks for all of the texts were examined.

These are the patterns that were used for evaluation:

1. **Noun-Verb (NV)** — Aims to capture sentences starting with a NP followed by a VP. The VP excludes conjugations of the verb *be*, so typically the NP is expected to be the *agent* in the sentence.

2. **What/Who matches a definition (MD)** — Aims to identify a definition (of the structure NP was|is|are|were NP) and generate a question regarding the NP with that definition.

3. **How something/someone is defined (HD)** — Similar to MD, but focuses about the definition itself.

4. **By who something was done (BW)** — Aims to capture structure of a predicate and generate a question about its agent. This is done by detecting a VP connecting two NPs, one of which following the preposition *by*.

5. **Time entity (Time)** — Generate gaps in the place of a time NE.

6. **Numeric entity (Num)** — Generate gaps in the place of a numeric NE.

7. **Location entity (Loc)** — Generate gaps in the place of a location NE.

As can be seen, some of the patterns (the first part) deal with the syntactic structure of the sentence and changes it in order to formulate a question, whereas the lower part tackles semantic targets and makes the learner directly recalling the semantic gap without
the need to deal with a transformed sentence.

Table 2 lists all of the patterns used for evaluation, along with an example of a sentence they would capture. All of the example tasks are MCQs. Those with the gaps are simply not formulated in a form of a question, but still offer the learner possible answer to choose from.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Sentence</th>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV</td>
<td>Chris drank an espresso</td>
<td>Who Drank an espresso?</td>
<td>Chris</td>
</tr>
<tr>
<td>MD</td>
<td>Mozart was a composer</td>
<td>Who was a composer?</td>
<td>Mozart</td>
</tr>
<tr>
<td>HD</td>
<td>Mozart was a composer</td>
<td>What was Mozart?</td>
<td>a composer</td>
</tr>
<tr>
<td>BW</td>
<td>Kennedy was killed by an assassin</td>
<td>By what Kennedy was killed?</td>
<td>an assassin</td>
</tr>
<tr>
<td>Time</td>
<td>Einstein received the Nobel Prize in 1921</td>
<td>Einstein received the Nobel Prize in ____</td>
<td>1921</td>
</tr>
<tr>
<td>Num</td>
<td>There are 50 states in the USA</td>
<td>There are ____ states in the USA</td>
<td>50</td>
</tr>
<tr>
<td>Loc</td>
<td>Sherlock Holmes works in London</td>
<td>Sherlock Holmes works in ____</td>
<td>London</td>
</tr>
</tbody>
</table>

Table 2: List of the pattern used for evaluation with example sentence, resulted question and its answer

4.3 Output Examples

Some of the tasks the evaluation yielded are particularly interesting and worth being presented and discussed.

To start from the successful portion of the examples, it is important to point out that the used pattern mostly succeeded in capturing the semantic meanings they were supposed to and targeted the correct part(s) of the sentence. An example for such a good result can be seen in examples (5) and (6), which are based on the same sentence. Both tasks are grammatical and relevant to the text, and all of their distractors are taken from the text itself (the correct answer is in bold script).
(5) Who received the Nobel Prize in Physics in 1921?
   a. Albert Einstein
   b. Elsa Lowenthal
   c. Ehud Olmert
   d. Mileva

(6) Albert Einstein received the Nobel Prize in Physics in ______.
   a. 1986
   b. 1919
   c. 1921
   d. 1940

These examples demonstrate one more aspect in task formulation, namely co-reference. In this case, of the name Albert Einstein in the sentence *He received the Nobel Prize in Physics in 1921*. This correct co-reference made it possible to use this sentence for a task, seeing that start a question with “He received […]” or presenting “He” as the correct answer for a question is not acceptable. Another example for such a successful transformation can be seen in example (7), where the co-reference points once again to *Albert Einstein*, which is also the correct answer.

(7) Who developed the general theory of relativity?

An interesting case is presented in example (8), where the task itself is valid, but the context of the gap reveals the correct answer due to the singular form of the following noun. It may be stated, however, that filling such a gap still shows some kind of basic language capabilities (for example, if this task is presented to a 2nd year English as a Foreign Language (EFL) learner who still needs to prove his/her ability to handle agreement) and only if just because it makes the learner read the task and the correct answer and by that
dealing with the material. Other distractors that didn’t match the gap’s content included, among others, fraction for filling a cardinal number counting people, objects as subjects for predicates only animated nouns could be agents of, names identified as location (and therefore selecting inappropriate distractors) and more.

Example (9) shows another valid task with problematic distractors. This time, choice a is marked as the correct answer, but choice d might be (and in fact it is, in this case) correct as well, seeing that Zurich is a city in Switzerland. Relying purely on the text itself, choice a is indeed the correct answer, but telling a learner who marked choice d that his/her answer is incorrect is unacceptable. Similar case was observed in a question where two distractors were practically identical, namely U.S. and Untied States. This happened due to insufficient co-referencing, since both of these choices should have been referenced to the same entity. Such issues are a result of external tools and could not be handled or modified directly by ARCQC.

(8) Conventionally, a computer consists of at least _____ processing element, […]
   a. 42
   b. one
   c. hundred
   d. 7

(9) When Albert Einstein became older, he went to a school in _____.
   a. Switzerland
   b. Princeton
   c. New Jersey
   d. Zurich

Some other transformations, however, weren’t so successful (for more details refer to table 4). For instance, example (10) shows a task that doesn’t make sense, since it doesn’t
refer to a whole sentence. This is an example of insufficiently defined pattern. On the bright side, most of the distractors did match the gap. One of the problems with news texts is direct speech. Direct speech is a more evolved structure, since it combines multiple sentences in one — the cited sentence(s) and the sentences presenting the citation. Example (11), for example, was generated based on such a sentence. Since the captures part included the pronoun I, which was then co-referenced to a NE, the question word Who is correct. However, having I as the correct answer is confusing and the whole task might look a bit odd. Another issue in this task is the distractors, which don’t match the content grammatically. This would have been, at least partially, solved if a correct co-referencing would have taken place.

This kind of sentences were responsible to the majority of the sentences under the non-sense error category (see table 4), especially in the Sports and Politics text categories.

(10) In year _____

(11) Who think it’s too late, said Steve Elmendorf, a Clinton, whose support for LGBT issues went back years before same-sex marriage was a hot political topic fan and former chair of the Victory Fund, a group that provides financial and political support to gay candidates?
   a. the gay community
   b. Clinton
   c. a primary race
   d. I

Finally, example (12) demonstrates probably the worst type of error, a valid task with valid distractors with an incorrect choice marked as the correct answer. The choice marked as correct looks — and could theoretically have been — the correct one, but it’s not (due to incorrect co-referencing). In this case, the correct answer (Einstein) does appear as one
of the choices, so that in a written quiz with a human inspector, this problem can be bypassed. Learners who read the text thoroughly may recall that Ehud Olmert was a prime minister of Israel, but if a computer is to check their answers, they will not get the points. While this error is somewhat amusing, it is also very problematic, because it misleads the learner by presenting incorrect information, and therefore might damage his/her learning process and motivation.

(12) Who taught physics at the Institute for Advanced Study at Princeton?
   a. Einstein (should be correct)
   b. Maja
   d. Ehud Olmert
   c. Elsa Lowenthal

There are many more examples of interesting successful and unsuccessful generations, but unfortunately discussing all of them would exceed the scope of this paper.

4.4 Results and Discussion

The evaluation data is summarized in table 3. Tasks were generated for each one of the texts in each of the chosen categories. Except for the Sports category, all of the texts were more or less of the same length, in average. The degree of validness of tasks and distractors is around 75%. So-called “syntactic-oriented tasks” and “semantic-oriented tasks” were almost equally generated (see figure 7 for detailed distribution).
Table 3: Evaluation data summary. On the left-hand side, averages per text in each category; in the middle the average distribution of each pattern per text; and to the right the validness of the generated tasks and distractors.

Deeper analysis of the generated tasks shows that the two main principles of ARCQC (see details in The System — ARCQC) were realized. First, only seldom was it required to acquire additional distractors from an external list (which was itself created based on previous texts, see Distractor Selection), and even then, they fit the other distractors and the context. Secondly, all of the patterns used were satisfyingly triggered (with the exception of the BW pattern, which was triggered only a few times, see figure 7) and may be further customized by anyone using the program.

One surprising observation is the relative similarity between the results of the Reuter News and the Wikipedia and Simple Wikipedia texts. The latter were expected to trigger

Table 4: Classification of error types in tasks and distractors generation

<table>
<thead>
<tr>
<th></th>
<th>Tasks</th>
<th>Distractors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ungrammatical</td>
<td>23%</td>
<td>33%</td>
</tr>
<tr>
<td>Non-Sense</td>
<td>56%</td>
<td>41%</td>
</tr>
<tr>
<td>Wrong Target</td>
<td>18%</td>
<td>16%</td>
</tr>
<tr>
<td>Other</td>
<td>3%</td>
<td>10%</td>
</tr>
</tbody>
</table>
the patterns more often and to have more valid distractors, due to their simpler syntactic structure. This might be due to the fact that over 50% of the tasks were generated by the “semantic” patterns, and among the “syntactic” patterns, the one that was by far the most dominant pattern, the NV pattern, captures a syntactic structure that is very likely to occur in any English text.

Another observation is that the BW pattern was the least triggered one. This may be ascribed to the fact that it captures the most complex structure of all of the patterns, which happened not to occur often in the chosen texts.
Figure 7: Distribution of the generated tasks for the entire data

As can be seen in the various Output Examples presented previously, there were multiple reasons for tasks to be classified as invalid: Ungrammatical (syntax-oriented), Non-Sense (semantic-oriented), Wrong Target (program/pattern-oriented) and other (none of the others). Table 4 shows how influential each of those reasons was.

Figure 7 shows the distribution of triggering of each of the used patterns.

To sum up, the evaluation done for ARCCQC demonstrates the feasibility of the approach and of the functionality of the customizable XML patterns. Its results show a high degree of validness of the generated tasks. However, some measures could be taken in order to improve them. First, more extensive normalization and some basic simplification measures (like apposition deletion, acronym expansion and other) would have made more sentences trigger the patterns. Second, better method for co-referencing and tokenizing would lead to fewer errors occur due to incorrect NEs and meaningless tokens (like possessive s, for example). Another mechanism that doesn’t currently exist and could improve the distractor selection process is matching the distractors to the target’s context based on deeper linguistic analysis. For instance, not allowing plural NP distractors to be selected for a task with a singular NP as answer. This can be done using dependency parsing and basic morphology analysis of the predicate on which the answer depends.
4.5 Performance

The Evaluation was performed on a dedicated server. Each category’s texts were processed as a batch (but not in parallel), so that all of the models needed to be loaded only once per category. In this constellation, it took about 70 seconds on average to analyze each category’s five texts. On an average personal laptop (e.g. with i5 processor and 8GB RAM) this process is estimated to take 30% – 40% longer. This time is considered to be reasonable and makes personal use of the tool feasible.

Since loading the models takes most of the processing time, processing texts in bundles and not separately is generally recommended.
5 Future Work

ARCQC was developed from scratch, during a limited time span, and in parallel to other course-works and projects. Therefore, it goes without saying that not all planned and desired features for it were implemented. Moreover, some ideas were not planned on to begin with, but can still be interesting additions to the system, providing the time and resources. This part of the paper will present some of these suggested future features and discuss their potential contributions to the system.

The first and most obvious improvement to the system would be the addition of more patterns. Since the goal of the project is to be able to generate questions about as many authentic texts as possible, all kinds of questions — with various foci — should be used. These additional XML patterns can be used to capture both very simple and more involved and sophisticated targets. Some ideas for such patterns were suggested throughout this paper, and there are more, including targeting more complex syntactic structure, combining all kinds of NEs to increase focus accuracy, formulating more natural and task-specific customized instructions for the tasks and more. The more patterns there are, the more tasks the system will be able to (at least statistically) generate for texts.

A relatively easy type of task to implement would be *X is a kind of task* (as introduced in Mitkov and Ha (2003); Mitkov et al. (2006)), where the semantic relation of a term is focused on. For example, *Lamborghini is a type of car* or *Mozart was a composer*. For this type of task a semantic-relation database is needed (e.g. WordNet\(^{19}\), Miller (1995)). However, the tricky part is not to break the self-contained generation principle (see Self-Contained Generation in The System — ARCQC). For this, all semantic relations will have to be extracted (and later verified by the semantics database) from the text itself. This might be hard to achieve with an authentic text, depending on its length, content,

\(^{19}\text{https://wordnet.princeton.edu/}\)
structure and other characteristics.

Directly connected to the quantity of the patterns is the quality of them. More specifically, the variety of linguistic features that is supported by the system and can be used in the patterns.

One of the features that was not included in this preliminary version of the system, and which could enrich the tasks generated by the system, is semantic functions. The system can currently capture the semantics of subject, predicate and object by defining their corresponding structures in a pattern and manipulating them accordingly. However, the addition of semantic function analysis (and of course a corresponding XML tag for it) will make it easier to capture the meaning of agent and patient, which might be more precise and direct in many cases and will save the engineering of syntactic structures in patterns. For example, such analysis can be used for focusing on interactions between two main figures in a story (together with NE conditioning). Tools like Standford CoreNLP offer some additional semantic analyses, which, after some computational work, can be translated into these semantic functions.

Another kind of focus for tasks, which goes beyond the sentence level, could be looking for common characteristics of terms (mostly noun-phrases, but not only) throughout the text. This can be used for asking questions of type “what is common to X, Y and Z”. For instance, if the sentences

(13) John called Mary
(14) John called Peter
(15) John called Mr. Smith

occur somewhere in the text, the question “what is common to Mary, Peter and Mr. Smith [in respect of John]?” can be generated, with the correct answer being “John called them”. This is of course a very simplistic example, but once the mechanism for such questions exists, it can be used to ask about time entities, location entities, relations
between main figures in a text, descriptions of terms (like adjective describing different objects in a text) and more. One of the challenges in implementing such a mechanism is deciding what information to store about each term (i.e. what characteristics can be targeted as common) and how (the best suited data structure, retrieval complexity etc.).

Two further possible improvements engage the graphical interface presented to the user. First, the quiz page generated for WERTi can be visually improved and matched to the standard layout of WERTi pages. Secondly, ARCQC can currently run either directly from a command-line terminal or indirectly from WERTi’s graphical interface. In order to make the system more independent, a built-in interface could be implemented for it. Through this interface, the user will be able to use all of the system’s features, including loading a text, generating a quiz, changing configurations, saving a quiz, loading and post-editing a quiz etc. Especially useful for the user would be a graphical interface for creating and editing patterns, without the need to have any knowledge about XML, which will allow people without technical backgrounds, such as teachers and other educational specialists, to experiment with the system and customize the quizzes to fit their educational goals and needs.

Last but not least, the feedback mechanism could be improved to be incremental. For example, instead of directly giving the full correct answer, the system could first give the learner a hint (like the sentence/line/paragraph where the answer can be found), and only upon request for further help would the full answer be given.

While this improvement may improve the learning process and make the learner engage a bit more with the text, it also breaks the homogeneous feedback mechanism of WERTi. However, since such an improvement must be implemented in the HTML/JavaScript output, the mechanism can just as well be implemented in other activities of WERTi (in a way that would be appropriate to them, of course, since they don’t have a separate text to rely on).
6 Summary

This paper introduced an end-to-end system for automatic re-call multiple-choice question generation about a given text. Customizable XML patterns determine what question will be generated. Among other things, these patterns contain information about the structure of the sentence to target, specific elements in it to occur, what part of the sentence will be marked as the correct answer, how to transform the target sentence into a task and more.

The system determines the main terms in the given text using a TF-IDF scoring mechanism. Another mechanism is responsible for filtering out questions that are considered invalid or not relevant enough. The distractors for the questions are almost exclusively selected from the text itself.

The accessible customization of the patterns and the focus purely on the given text itself make the system suitable for reading-check and demonstration of at least shallow comprehension of contents in the text. The resulting tasks can be displayed either as editable plain text or in an interactive web-page.

An evaluation of the system was carried out to demonstrate the feasibility of the approach. 20 Reuters News articles and 10 Wikipedia entries were processed by the system and quizzes (sets of tasks) were generated for them, based on seven patterns created especially for this evaluation. The results show that, in average, about five tasks were generated for every 100 words in the texts. Furthermore, 74% of the tasks and 77% of the distractors were found to be valid (i.e. grammatical, relevant, correctly formulated etc.). The main errors that caused tasks and distractors to be invalid were incorrect co-referencing, problematic tokenizing or chunking, linguistic disagreement between gap/target and distractors, and over-detection of patterns.

Improvement opportunities for the system include broadening the support for linguistic structures to include more types of targets, generating tasks based on structures beyond the sentence level, enhancing the feedback given in the interactive web-pages and more.
A Bibliography

References


Anita Ferreira, Johanna D Moore, and Chris Mellish. A study of feedback strategies in foreign language classrooms and tutorials with implications for intelligent computer-


B  List of Acronyms

**AQG** Automatic Question Generation

**ARCQC** Automatic Re-Call Questions Creator

**CL** Computational Linguistics

**EFL** English as a Foreign Language

**FIB** Fill in the Blank

**ICALL** Intelligent Computer-Assisted Language Learning

**ILTS** Intelligent Language Tutoring System

**ITS** Intelligent Tutoring System

**MCQ** Multiple Choice Question

**NE** Named Entity

**NLP** Natural Language Processing

**POS** Part of Speech

**RC** Reading Comprehension

**RQ** Recall Question

**SLA** Second Language Acquisition

**TOEFL** Test Of English as a Foreign Language

**TOEIC** Test of English for International Communication

**TF-IDF** Term Frequency — Inverted Document Frequency

**WERTi** Working with English Real Text (interactive)

**XML** eXtensible Markup Language