Measuring the Quality of Assessment Using Questions Generated from the Semantic Web

Ricardo Conejo(✉), Beatriz Barros, and Manuel F. Bertoa

University of Malaga, Malaga, Spain
conejo@lcc.uma.es

Abstract. This article describes a new feature of the adaptive assessment system SIETTE that allows for the static and dynamic generation of questions from tables of data for knowledge assessment. Almost the same approach can be used to generate questions from data collected in a spreadsheet, a database query, or a semantic web query using SPARQL. The main problem faced with question generation is ensuring that the questions are valid for assessment. For this reason, most of the existing systems propose to use this mechanism only for low-stakes assessments. In this paper, we propose a methodology to control question generation quality and measure the impact of potential invalid instances on the final score as well as recommend some strategies to overcome these problems.

Keywords: Question generation · Semantic web
Automatic assessment

1 Introduction

Question generation from databases [1], and questions from linked open data (LOD), in particular, are a potential source of an unlimited number of questions for a wide variety of uses. For example, Le et al. [2] distinguish between knowledge/skills acquisition, tutorial dialogues, and knowledge assessment.

This article focus on question generation for knowledge assessment, especially when the questions are generated from semantic web sources, like Wikidata or DBPedia. There are already a number of systems that have identified this potential use [3–5]. However, the messy structure and the number of invalid instances that are generated among the valid ones, implies that most of these systems only aim at recreational or self-assessment use [6–8].

We face the problem of using generated questions for high-stakes knowledge assessment in SIETTE, defining a methodology to reduce the number of incorrect instances, measuring the impact of potential invalid instances on the assessment score, and defining some strategies to correct and reduce these problems.

C. Penstein Rosé et al. (Eds.): AIED 2018, LNAI 10947, pp. 57–69, 2018.
https://doi.org/10.1007/978-3-319-93843-1_5

2 Question Generation in SIETTE

SIETTE is a general-purpose, domain-independent assessment environment (see [9,10]). The system incorporates item banking, test building, analysis, delivery and result presentation. It supports classical test theory (CTT), item response theory (IRT), and computer-adaptive testing (CAT). Three main types of item models are supported: (1) multiple choice, single answer questions (MCQ-SA); (2) Multiple choice, multiple answer questions (MCQ-MA); and (3) short answer questions (SAQ). These are just the three internal models that SIETTE uses. Any other type of question that SIETTE can deal with (like sorting, correspondence, drag and drop, etc.) are represented by one of the item models.

The first two item models are simpler to evaluate, but the third one requires an engine that recognizes the student's answer and assigns it into a given pattern solution. SIETTE provides different types of patterns for different uses. The most commonly used are regular expression patterns. For instance, a question could be "*Who is the composer of the Moonlight Sonata?*". The pattern "`{Ludwig {van}} Beethoven`" will accept answers like *Beethoven* or *Ludwig van Beethoven*. Multiple patterns can be used for the same question. Additionally, at a teacher's option, patterns can ignore case, accented characters, white spaces, and/or punctuation symbols. They can also includes numerical ranges, magnitudes, and more. If the response of a question is a number, SIETTE allows the construction of patterns that include a variable range. For instance, the SIETTE pattern "[3,14%1]" will accept any number close to π, and the SIETTE pattern [1450#1500] will accept any number between this range. Magnitudes can be added to numerical ranges to recognize different expressions, meaning that [72 km/h%1] will also recognize 20 m/s. Questions might have several correct answer patterns, and even patterns to recognize common incorrect answers, in order to provide accurate feedback.

In some ways, it is easier to generate good SAQs than good MCQs. For the first type, it is necessary to generate a pattern that recognizes the response, which is very easy with numerical values and single-word text. In the second case the main difficulty is to generate alternative options (often called distractors) that are plausible. We will revisit this issue later.

On the other hand, SIETTE can generate questions from templates written in JSP, or any other server-side script language. For instance, a question template that generates two numbers between 0 and 5 and asks for their sum could be:

```
<%
   int x       = Math.abs( siette.util.Random.nextInt() % 5 ) + 1;
   int y       = Math.abs( siette.util.Random.nextInt() % 5 ) + 1;
   int z       = x+y;
%>
What is the result of this operation:<br/>
<center> <%= x %> + <%= y %> </center>
```

An important detail is that SIETTE uses its own *Random* class, that is fixed by setting a seed, for each question according to the question and assessment session id, so the template will always generate the same random instance each time it is called by the same test session. This is important because: (1) there is no need to save the instance itself, just the question template and the seed; and (2) each time the question is posed or reviewed in a test session, the same instance is generated, allowing a user to go back and forth during an assessment. It also allows the teachers to review the assessment session without changing the originally posed instance. On the other hand, this approach has the drawback that if the structure of the question template is changed after the student responds, the instances might differ. However, this is easily solved by locking the question templates once they are posed.

In the simpler cases, the templates generates random numbers or strings that are inserted in the question. A first extension of this technique is the generation of questions from tables containing a data set. To illustrate this implementation, we introduce the example of the periodic table. Suppose we have a spreadsheet containing 103 rows corresponding to the first 103 chemical elements. Columns are labeled NAME, SYMBOL, ATOMIC NUMBER, and so on. From this spreadsheet, it is possible to generate multiple questions, for instance, the symbol of a given element:

```
<%@page import="siette.util.corpus.Table"%>
<%
   Table table = new Table("demo/periodic-table.xls");
   String[] element = table.select();
   String name = table.get( element, "NAME" );
   String symbol = table.get( element, "SYMBOL" );
%>
What is the symbol for the chemical element "<i><%= name %></i>"?
```

There are many other possible question templates that can be generated from the same table. The easiest way is to generate SAQs, but other types are also possible, like *Which of these three elements has the lowest atomic number?* (MCQ-SA question) or *Select from among these six, all the elements that have a density greater than lead* (MCQ-MA question).

The following extension is to read data from a database, rather than from a spreadsheet. As an example, we have used the database from project TREE [11], that contains 32 tables with information about 322 European tree species, including images labeled with metadata that describes them.

```
<%@page import="siette.util.corpus.DatabaseTable"%>
<%
  String query = "SELECT S.BINOMIAL, S.COMMON_NAME, P.IMG "
               + " FROM SPECIES S JOIN PHOTOS P ON P.SPECIE = S.SPECIE "
               + " WHERE P.FEATURE='Leaf' "
               ;
  DatabaseTable table = new DatabaseTable("demo/tree.properties", query);
  String[] plant = table.select();
  String img = table.get( plant, "IMG" );
  String binomial = table.get( plant, "BINOMIAL" );
  String plant_name = table.get( plant, "NOMBRE" );
%>
Write the scientific name of the specie that have this leaf:
<center> <IMG SRC='<%= img %>'> </center>
```

In this case, the accepted pattern might be the binomial scientific name or the common name, although in this case the query should be more complex to cover all possible responses.

The next step is the extension to query from the semantic web using SPARQL. The idea is similar to the previous implementation, but the data source is broader. As an example, the following query from DBpedia generates questions about the flag of a given country.

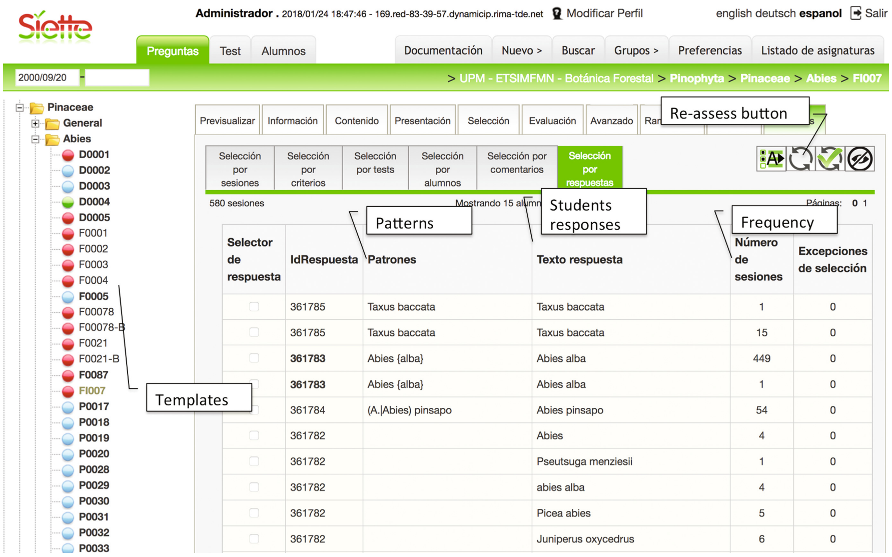
```
<%@page import="siette.util.corpus.WebTable"%>
<%
String query = "SELECT DISTINCT ?nombre ?name ?population ?flag ?img "
+"WHERE { "
+"    ?country a dbpedia-owl:Country ; rdfs:label ?name ; dbo:flag ?flag . "
+"    ?country ?hasPopulation ?population ; dbo:thumbnail ?img . "
+"    ?country dct:subject dbc:Member_states_of_the_United_Nations "
+"    FILTER (langMatches(lang(?name), \"en\")) "
+"    FILTER (?population > 1000000)  "
+"} "
;
   WebTable table = new WebTable("http://dbpedia.org/sparql",query);
   String[] country = table.select();
   String name = table.get( country, "name" );
   String img  =  table.get( country, "img" );
%>
<center>
Which is the country of this flag? <br/>
<center><IMG src="<%= img %>"/><br/></center>
```

Notice that the query has restricted the set of countries to those that are members of the United Nations, and have more that 1 million inhabitants.

In the same way, it is possible to generate questions like *Who is the painter of this picture? (+image)* or *Who is the composer of this musical work? (+sound)*. Most of the current approaches to question generation from the Semantic Web generate MCQs [4,12,13]. This approach makes it easier to recognize the correct student response, but it makes it much more difficult to generate appropriate questions. The key point is that options should be plausible. To accomplish this, these systems define a distance between possible responses based on the amount of metadata that matches both the correct answer and the alternatives and other ontology distance measures. Practical results are not always satisfactory. Moreover, generating SAQs usually involves just a single record from the database, while generating multiple choices involves multiple records. Reducing the number of records decrease the potential problems for generating an invalid question (see next section). Nevertheless, deep question generation may involves multiple records and relations [14,15].

Additionally, it is possible to generate hints or feedbacks by selecting some important feature and displaying one or two of the first sentences from the `rdfs:comment` field. For instance, in a case of where the correct answer is Beethoven, the hint could be ___ *was a German composer and pianist. A crucial figure in the transition between the Classical and Romantic eras in Western music art; he remains one of the most famous and influential of all composers.* Of course, this requires removing any text that matches the correct response pattern. Later paragraphs might give less obvious hints, like *His best known compositions include 9 symphonies, 5 piano concertos, 1 violin concerto, 32 piano sonatas, 16 string* *quartets, his great Mass the Missa solemnis, and 1 opera.* Another way to generate hints is by selecting other similar records. For instance, if we are asking for a composition by Beethoven, a hint could be *This other piece was composed by the same musician (+sound)*, which is easily obtained with a complementary query.

It is not possible to guarantee that the pattern will match 100% of students' responses. However, SIETTE includes a simple mechanism that displays all student responses on a single page, allowing for modification and reassessment of the pattern (see Fig. 1). This final step achieves 100% correction of actual student's responses, but implies an assessment review, which as we will see, is also necessary when dealing with invalid instances (see Subsect. 4.2)



Selector de respuesta	IdRespuesta	Patrones	Texto respuesta	Número de sesiones	Excepciones de selección
	361785	Taxus baccata	Taxus baccata	1	0
	361785	Taxus baccata	Taxus baccata	15	0
	361783	Abies {alba}	Abies alba	449	0
	361783	Abies {alba}	Abies alba	1	0
	361784	(A.|Abies) pinsapo	Abies pinsapo	54	0
	361782		Abies	4	0
	361782		Pseutsuga menziesii	1	0
	361782		abies alba	4	0
	361782		Picea abies	5	0
	361782		Juniperus oxycedrus	6	0

Fig. 1. Listing of all patterns and students' answers for a botany question

3 Problems with Generated Questions

The previous section outlined the technicalities involved in question generation from tables. However, the main problem is not generating the question but ensuring its quality and validity. The technical procedure is the same in the three cases, but there are differences between using a spreadsheet, database or semantic web data. In the first case, the table is uploaded by the teacher to the SIETTE environment. It is commonly constructed for assessment purpose, has a relatively low fixed number of rows, and can be edited by the teacher in case a mistake is detected. In the second case, the database is bigger and the data are mostly correct, but it might not be under the teacher's control. In the last case, the database is huge but there are a significant number of potential problems with data that are not correctly labeled.

Of course, a trivial solution for the problem of generating questions from uncontrolled sources is to create local dumps, that is, generate a spreadsheet or a database from a semantic web query and manually review each record. Another trivial solution (which is also implemented in SIETTE) is to produce a set of static (fixed) instances from a template question, add them to the item bank as normal questions, and review them one by one. These trivial approaches also have its drawbacks because the dynamic behavior is lost, and future changes in the database do not affect the generated questions. Moreover, it might require larger space to store the generated questions, with possible implications for system efficiency. Consider a small number of queries that produce a large number of instances. In this case, the item bank will be filled up with several copies of similar questions. This fact might reduce, for instance, the efficiency of adaptive item selection process. On the other hand, manually reviewing of a huge number of generated questions would be a very tedious task.

In the rest of the paper, we will focus on the problem of dynamic question generation using semantic web queries, which is the most challenging feature. Techniques for detecting and correcting low-quality items can be applied in the three cases, but there are different sources of problems. As a result, we face different problems when questions are generated from semantic web queries, like those detailed below:

1. The first problem is that the databases might not be completely correctly labeled. We have tried DBpedia data with toy examples like the composer of a musical work and found some mislabeled pieces. However, this source of problems is rare.
2. Another source of problems is missing content. For instance, when we tried the query about flags described in the previous section, the link to the Swiss flag was broken. This problem can be avoided at run time by checking the link previously and generating a different instance if there is a broken link.
3. A third type of failure is related to "undesired" or "unexpected" content. This happens, for instance, with a first version of the flags question, when the query in the template generated a question for the flag of *Prussia* or the flag of *Faeroe Islands* which is, in fact a country within Denmark, or for the flag of *Nuaru*, which is a very small island in Micronesia.
4. Another source of problems is the incompleteness of the response pattern in some particular cases. This occurs when the general procedure of constructing the pattern fails for a given instance. For instance, the pattern {`United States of`} `America` which is constructed for country names does not recognize the simpler answer *USA*. This type of problems can be avoided by carefully refining the SPARQL query and adding some alternative patterns by using redirection and/or disambiguating properties (for instance, by using the property `skos:altLabel` to gives name variations). The debug of this problem is done by random approaches, which is generating random test cases until a wrong case appears, correcting the query in that case, and then continuing with the debugging. However, this procedure does not guarantee complete success.

4 Measuring the Quality of Assessment

Depending on the desired use of the question templates, problems that arise can be more or less important, and the procedures to correct them should be more or less sophisticated. Suppose there is a test with n questions coming from n different templates that comes from queries that generating valid instances with a probability p. Further, the test is used for fun or for self-assessment. For high values of p (i.e., for $p > 0.90$) there is no need for further refinement. The user will notice the missing content and skip that question without further consequences.

If the question belongs to a high-stakes assessment, a similar precision might not be acceptable, because for a large population of students, there would be cases in which a relevant number of instance questions will be invalid, and the final score may be affected. In this case, we have to consider not just the average case, but the worst case that might occur among the total number of students. What differentiates these cases is the concept known as “test and score reliability”. Informally speaking, test reliability refers to the degree to which a test is consistent and stable in measuring what it is intended to measure. Score reliability refers to the confidence interval in which the true score of a subject is included.

CTT has proposed several definitions of test reliability [16], and one of the most commonly used is based on the concept of a parallel test, meaning that if a test is repeated with a similar question composition, the results should be the same. There are many statistics proposed to estimate the internal consistency reliability, commonly shortened as $\rho_{xx'}$. The most familiar are the Spearman-Brown, Guttman-Flannagan's λ_4, Kuder-Richardson's KR-20 or KR-21, and Cronbach's α. Test reliability varies from -1 to 1. Even in the most carefully manually prepared test, test reliability is not 1. Values over 0.70 are considered good enough.

The true score confidence interval can be estimated assuming normal error distribution as $X \pm z_{\alpha/2}\sigma_x\sqrt{1-\rho_{xx'}}$, where X is the score obtained in the test, α is the confidence level, and σ_x^2 is the variance of the whole test. If a test contains an invalid question, the student will always fail it, so test reliability will be reduced and variance will increase and, likewise, the interval will also increase.

However, if the number of incorrect questions is small compared to the number of questions in the test, the final reliability might not be compromised.

The research interest at this point is to be aware of the advantages and drawbacks of question generation, quantify the problem, and establish limits of acceptance as in any other engineering activity, implementing quality control procedures previous to the application of the test (see Sect. 4.1). Fortunately, even in high-stakes exams, there are no irremediable consequences, and an assessment can be reviewed. We are also interested in procedures to automatically or semi-automatically detect failures, correct them and reassess (see Sect. 4.2).

4.1 Preliminary Question Analysis

Question generation from huge databases is a kind of industrial production process, and thus, the same quality measurements should be taken. We distinguish two separate phases: software testing and statistical testing.

First, the template query should be proved and refined as much as possible to avoid undesirable results. We call this phase software testing, because software modification techniques are used to improve the question quality. However, as was exemplified in the previous section, query refinement reduces but does not eliminate all potential errors. Moreover, the successive refinements might introduce new invalid instances or impose unnecessary limits on potential instances candidates. To sum up, there is a limit in what can be obtained with software testing.

The first step of the statistical testing phase is to establish a desired confidence level and quantify the expected errors. As was suggested in the previous section, confidence level depends on the assessment goal. Suppose a test with n questions coming from n templates that comes from queries where each one generates a valid instance with probability p, the probability of having a maximum of w wrong instances in the test can be obtained by the cumulative binomial distribution

$$p_w = Pr(X \leq w) = \sum_{i=0}^{w} \binom{n}{i} p^n (1-p)^{n-i}$$

If m students take the test, the probability of having all sessions with less than w wrong instances is p_w^m.

In the next section, we will see that wrong instances can be removed and the test can be reassessed, but there would be a side effect in test reliability, so it is desirable to keep w below the acceptable limits, thus ensuring that p, which is not known, is below a threshold.

Proceeding in the opposite direction, assuming the value of w is desired to be less than a given number (fixed according to the maximum acceptable proportion of wrong instances in any test), the confidence level would be p_w for a single session and p_w^m the confidence level that it is not going to occur in any of the m test sessions. For instance, in a test on $n = 30$ questions, the probability of that any of the $m = 250$ includes less than $w = 3$ invalid instances (less than the 10%) would be $p_w^m \geq 0.945$ if the probability of a single failure is $p \leq 0.01$. Another way of viewing the situation is to predict the number of sessions with w errors. In this case, 73.9% of the cases will have no incorrect instance; 22.4% will include just one failure; 3.2% will include 2 failures; 0.3% will include 3 failures; and only 0.0012% sessions will include 4 or more.

4.2 Posterior Question Analysis

Achieving 100% accuracy in all cases is almost impossible, even with manually generated static questions. For this reason, SIETTE includes a review procedure that allows for editing of the assessment setting and reassessing all assessment sessions according to the given responses. The procedure is easy and can be done with a few clicks.

This procedure is initiated if the teacher or any student detects a failure in the test and request for an assessment review. To facilitate this task, SIETTE can optionally include a comment box beneath each question to collect during-test comments from the student for the posed question. All student comments are displayed together for the same question, so the teacher can evaluate if that question has some kind of misbehavior. There is also built-in automatic detection of incorrect questions that will be explained later. There are two possible cases of incorrect questions: (1) questions with a correct answer different from the one that is set for MCQ, or questions that have an incorrect or incomplete responses pattern for SAQ; and (2) questions that have a problem with the stem and could not be answered correctly. In the first case, the solution is to change the correct answer to the right one or provide a new or better pattern, but in the second, the only remedy is to remove that question from the assessment and reassess according to the remaining $(n-k)$ questions, supposing k items should be removed. The reduction in test reliability can be obtained by applying the Spearman-Brown prophecy formula

$$\rho^*_{xx'} = \frac{R\rho_{xx'}}{1+(R-1)\rho_{xx'}},$$

where R is the ratio between assessment lengths, that is, $n/(n-k)$. Table 1 shows the impact on reliability according to the percentage of items lost. Table 2 shows the percentage of variation of the confidence interval.

Table 1. New reliability $\rho^*_{xx'}$ as a function of original reliability and percentage of invalid questions

$\rho_{xx'}$	50%	40%	30%	20%	10%	5%	2.5%
0.70	0.538	0.583	0.620	0.651	0.677	0.689	0.696
0.75	0.600	0.643	0.677	0.706	0.730	0.740	0.746
0.80	0.667	0.706	0.737	0.762	0.783	0.792	0.797
0.85	0.739	0.773	0.799	0.819	0.836	0.843	0.847
0,90	0.818	0.844	0.863	0.878	0.890	0.895	0.898
0,95	0.905	0.919	0.930	0.938	0.945	0.948	0.949

A reduction in reliability implies that the standard error increases, and so does the confidence interval for a given confidence level α. The variation is given by the ratio $\sqrt{1-\rho^*_{xx'}}/\sqrt{1-\rho_{xx'}}$. Table 2 indicates the ratio for different combinations of original reliability and percentage of items lost. According to these tables, the number of test items, and the assessment application, it is possible to decide which is the maximum number of items that eventually might be invalidated. For instance, if an assessment is composed of 30 items, and has a reliability of 0.80, removing three items will imply that the confidence interval of the final score will increase 4.3%. Of course, the decision can be the other way around, that is, increasing the assessment length 10% (see [16] for further details about reliability and quality of test scores).

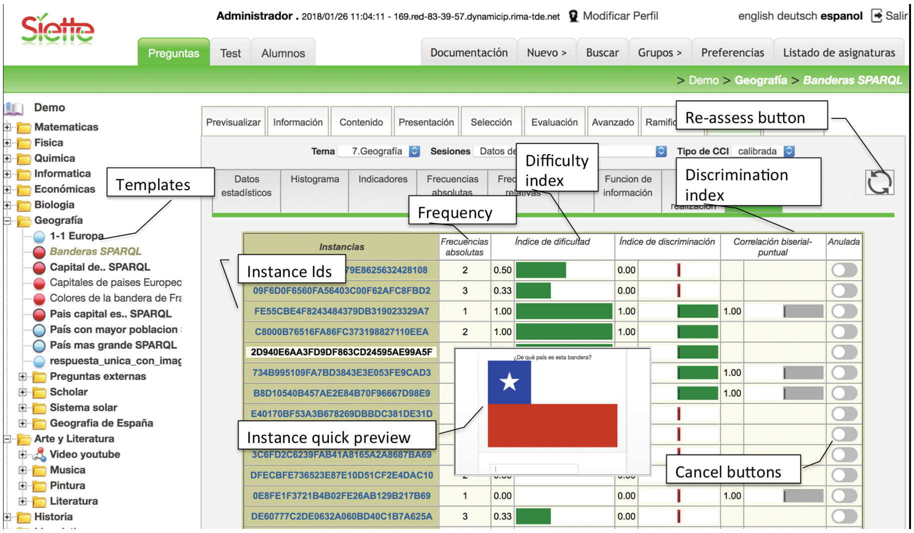
Table 2. Ratio of standard errors before and after removing a given percentage of questions

$\rho_{xx'}$	50%	40%	30%	20%	10%	5%	2.5%
0.70	1.240	1.179	1.125	1.078	1.037	1.018	1.007
0.75	1.265	1.195	1.136	1.085	1.040	1.019	1.008
0.80	1.219	1.213	1.147	1.091	1.043	1.021	1.008
0.85	1.319	1.231	1.159	1.098	1.045	1.022	1.009
0,90	1.348	1.250	1.170	1.104	1.048	1.023	1.009
0,95	1.380	1.270	1.183	1.111	1.051	1.025	1.010

The detection of incorrect instances can be done automatically using psychometric item analysis tools that are already integrated in SIETTE. There are three simple measures that allow for early detection of invalid instances. The *difficulty index* is defined as the proportion of students who answered the item correctly $p = \frac{C}{N}$ where C is the number of student that have answered the item correctly, and N is the total number of students. The *discrimination index* is defined as the difference between the ratio of hits of the first (p_{1q}) and fourth quartiles (p_{4q}). That is, the whole sample is divided into four parts according to the final score obtained in the assessment. The difficulty index is computed for the highest and lowest score parts, and the discrimination index is the difference between both: $D = p_{4q} - p_{1q}$. The point biserial correlation between the response and the item's total score is a good indicator of the item quality. IRT parameters can also be used for instance analysis.

An invalid instance will have a difficulty index and a discrimination index equal to zero (or very close to zero if we consider random responses). The discrimination index will help to identify not only invalid instances, but also those instances that are not valid for assessment, meaning those that do not relate to the knowledge we are measuring with the assessment.

Once the invalid items or instances have been detected, they are marked as *canceled* in the item pool. The test session is reassessed, and those questions marked as *canceled* are considered to have not been posed in every session where they might have appeared. Notice that if the percentage of invalid questions generated by a given template is small, this process will still significantly reduce that figure because the instances marked as *canceled* will be avoided in future assessment sessions. Figure 2 shows the indicators obtained for all the instances of a given template, the cancel slider and the reassess button.



Instancias	Frecuencias absolutas	Índice de dificultad	Índice de discriminación	Correlación biserial-puntual	Anulada
79E8625632428108	2	0.50	0.00		
09F6D0F6560FA56403C00F62AFC8FBD2	3	0.33	0.00		
FE55CBE4F8243484379DB319023329A7	1	1.00	1.00	1.00	
C8000B76516FA86FC373198827110EEA	2	1.00	1.00		
2D940E6AA3FD9DF863CD24595AE99A5F					
734B995109FA7BD3843E3E053FE9CAD3				1.00	
B8D10540B457AE2E84B70F96667D98E9				1.00	
E40170BF53A3B678269DBBDC381DE31D					
3C6FD2C6239FAB41A8165A2A8687BA69					
DFECBFE736523E87E10D51CF2E4DAC10					
0E8FE1F3721B4B02FE26AB129B217B69	1	0.00	0.00	1.00	
DE60777C2DE0632A060BD40C1B7A625A	3	0.33	0.00		

Fig. 2. Listing all instances' indicators to allow canceling of incorrect instances

Measuring difficulty and discrimination indexes is just one of many easy ways to control item quality. SIETTE also includes IRT procedures to calibrate the item bank, obtaining the IRT parameters. To use them, it is also a good idea to use anchor items, which are items that appears in every session and that can be used to establish a common ground.

The dynamic analysis might also suggest that the question generation template produces items with very different IRT item parameters; that is, instances generated by the same template that behave differently. In this case, the best strategy is to split the original template into two or more templates to guarantee fully isomorphic items.

5 Conclusions

Using public databases and the semantic web as source for question generation has a high use potential. However, current applications are restricted to low-stakes assessments. In order to use them for high-stakes assessments, we should be aware of for implications and establish quality control measures and review procedures to ensure the assessment validity and reliability. Following these rules, it is perfectly possible to use this immense source of knowledge for any type of assessment. We plan to apply this technique to generate questions for botany courses at Madrid Polytechnic University.

To sum up, the conclusions of this article can be summarized as follow: (1) generating SAQs reduces the number of invalid questions; (2) invalid questions can be detected manually or automatically by using item analysis techniques; (3) canceling invalid questions and re-assessing guarantees a valid assessment; and

(4) the effect of using invalid questions can be measured and acceptance limits can be established. We have implemented extensions of the SIETTE authoring tool that make it easier to accomplish these tasks. The system can be accessed at http://www.siette.org, where some sample tests, in Spanish and English, can be found in the Demo area.

References

1. Chaudhri, V.K., Clark, P.E., Overholtzer, A., Spaulding, A.: Question generation from a knowledge base. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) EKAW 2014. LNCS (LNAI), vol. 8876, pp. 54–65. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13704-9_5
2. Le, N.-T., Kojiri, T., Pinkwart, N.: Automatic question generation for educational applications – the state of art. In: van Do, T., Thi, H.A.L., Nguyen, N.T. (eds.) Advanced Computational Methods for Knowledge Engineering. AISC, vol. 282, pp. 325–338. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06569-4_24
3. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.C., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: Proceedings of the 21st International Conference on World Wide Web, WWW 2012, pp. 639. ACM Press, New York (2012)
4. Tamura, Y., Takase, Y., Hayashi, Y., Nakano, Y.I.: Generating quizzes for history learning based on wikipedia articles. In: Zaphiris, P., Ioannou, A. (eds.) LCT 2015. LNCS, vol. 9192, pp. 337–346. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20609-7_32
5. Foulonneau, M., Ras, E.: Using educational domain models for automatic item generation beyond factual knowledge assessment. In: Hernández-Leo, D., Ley, T., Klamma, R., Harrer, A. (eds.) EC-TEL 2013. LNCS, vol. 8095, pp. 442–447. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40814-4_36
6. Cablé, B., Guin, N., Lefevre, M.: An authoring tool for semi-automatic generation of self-assessment exercises. In: Lane, H.C., Yacef, K., Mostow, J., Pavlik, P. (eds.) AIED 2013. LNCS (LNAI), vol. 7926, pp. 679–682. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39112-5_87
7. Bühmann, L., Usbeck, R., Ngonga Ngomo, A.-C.: ASSESS — automatic self-assessment using linked data. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 76–89. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_5
8. Vega-Gorgojo, G.: Clover Quiz: a trivia game powered by DBpedia. Semantic-Web-Journal.Net
9. Conejo, R., Guzmán, E., Millán, E., Trella, M., Pérez-de, J.L.: SIETTE : a web based tool for adaptive testing. Int. J. Artif. Intell. Educ. **14**(1), 1–33 (2004)
10. Conejo, R., Guzmán, E., Trella, M.: The SIETTE automatic assessment environment. Int. J. Artif. Intell. Educ. **26**(1), 270–292 (2016)
11. Rios, A., Millán, E., Trella, M., Pérez-de-la Cruz, J.L., Conejo, R.: Internet based evaluation system. In: Artificial Intelligence in Education, vol. 64 pp. 1896–1898 (1999)
12. Bongir, A., Attar, V., Janardhanan, R.: Automated quiz generator. Adv. Intell. Syst. Comput. **683**, 174–188 (2018)

13. Alsubait, T., Parsia, B., Sattler, U.: Generating multiple choice questions from ontologies: how far can we go? In: Lambrix, P., et al. (eds.) EKAW 2014. LNCS (LNAI), vol. 8982, pp. 66–79. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17966-7_7
14. Jouault, C., Seta, K.: Content-dependent question generation for history learning in semantic open learning space. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 300–305. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07221-0_37
15. Olney, A., Graesser, A., Person, N.: Question generation from concept maps. Dialogue Discourse **3**(2), 75–99 (2012)
16. Wainer, H., Thissen, D.: How is reliability related to the quality of test scores? what is the effect of local dependence on reliability? Educ. Measure. Issues Pract. **15**(1), 22–29 (1996)