1 Which Assessment Questions are the Most Difficult

As learners engage in activities supported by a learning ecosystem, they will experience learning content as well as assessment content. Assessments are designed to measure the effectiveness of learning content and help assess knowledge gained. It is possible that certain assessment questions do not accurately represent the concepts contained within learning content and this may be indicated by a majority of learners getting the question wrong. It is also possible that the question accurately represents the learning content but is very difficult. The following algorithm will identify these types of questions but will not be able to deduce why learners answer them incorrectly.

1.1 Ideal Statements

In order to accurately determine which assessment questions are the most dificult, there are a few requirements of the data produced by a LRP. They are as follows:

- statements describing a learner answering a question must report if the learner got the question correct or incorrect via $\$.result.success$
- if it is possible to get partial credit on a question, the amount of credit should be reported within the statement
  - the credit earned by the learner should be reported within $\$.result.score.raw$
  - the minimum and maximum possible credit amount should be reported within $\$.result.score.min$ and $\$.result.score.max$ respectively
- If it is possible to get partial credit on a question, it must still be reported if the learner reached the threshold of success via $\$.result.success$
- Statements describing a learner answering a question should contain activities of the type *cmi.interaction*
- activities must be uniquely and consistently identified across all statements
- Statements describing a learner answering a question should[1] use the verb $http://adlnet.gov/expapi/verbs/answered$

[1] it is possible to use another verb iri but if another is used, that will need to be accounted for in data retrieval

1.2 Input Data Retrieval

How to query an LRS via a GET request to the Statements Resource via curl. The following section contains the appropriate parameters with example values as well as the curl command necessary for making the request.[2][3][4]

```
Verb = "verb=http://adlnet.gov/expapi/verbs/answered"

Since = "since=2018-07-20T12:08:47Z"

Until = "until=2018-07-21T12:08:47Z"

Base = "https://example.endpoint/statements?"

endpoint = Base + Verb + "&" + Since + "&" + Until

Auth = Hash generated from basic auth

S = curl -X GET -H "Authorization: Auth"
-H "Content-Type: application/json"
-H "X-Experience-API-Version: 1.0.3"
Endpoint
```

1.3 Statement Parameters to Utilize

The statement parameter locations here are written in JSONPath. This notation is also compatable with the xAPI Z notation due to the defined hierarchy of components. Within the Z specifications, a variable name will be used instead of the $

- $.*result.success*
- $.*object.id*

1.4 2018 Pilot TLA Statement Problems

The initial pilot test data supports this algorithm. Given that the offical 2018 pilot test is scheduled to take place on July 27th, 2018, this section may require updates pending future data review.

1.5 Summary

1. Query an LRS via a GET request to the statements endpoint using the parameters verb, since and until

[2] See footnote 1.
[3] See footnote 2.
[4] See footnote 3.

2. Filter the results to the set of statements where:
    - $\$.result.success$ is false
3. process the filtered data
    - group by $\$.object.id$
    - determine the count of each group
    - create a collection of pairs = [$\$.object.id$, #]

1.6 Formal Specification

1.6.1 Basic Types

$INCORRECT :== \{false\}$

1.6.2 System State

$MostDifficultAssessmentQuestions$

$Statements$

$S_{all} : \mathbb{F}_1$

$S_{incorrect}, S_{grouped}, S_{processed} : \mathbb{F}$

$S_{all} = statements$

$S_{incorrect} \subseteq S_{all}$

$S_{grouped} = \{groups : \text{seq}_1\ statement\}$

$S_{processed} = \{pair : (id, \mathbb{N})\}$

- The set S_{all} is a non-empty, finite set and is the component *statements*
- The sets $S_{incorrect}$, $S_{grouped}$ and $S_{processed}$ are all finite sets
- the set $S_{incorrect}$ is a subset of S_{all} which may contain every value within S_{all}
- the set $S_{grouped}$ is a finite set of objects *groups* which are non-empty, finite sequences of the component *statement*
- the set $S_{processed}$ is a finite set of pairs where each contains the component *id* and a natural number

1.6.3 Initial System State

$InitMostDifficultAssessmentQuestions$

$MostDifficultAssessmentQuestions$

$S_{all} \neq \emptyset$

$S_{incorrect} = \emptyset$

$S_{grouped} = \emptyset$

$S_{processed} = \emptyset$

- The set S_{all} is a non-empty set
- The sets $S_{incorrect}$, $S_{grouped}$ and $S_{processed}$ are all initially empty

1.6.4 Filter for Incorrect

$$
\begin{array}{|l}
\hline
Incorrect \\
\hline
Statement \\
incorrect : STATEMENT \pfun \mathbb{F} \\
s? : STATEMENT \\
s! : \mathbb{F} \\
\hline
s? = statement \\
s! = incorrect(s?) \\
incorrect(s?) = \textbf{if } s?.result.success : INCORRECT \\
\qquad\qquad\qquad \textbf{then } s! = s? \\
\qquad\qquad\qquad \textbf{else } s! = \emptyset \\
\hline
\end{array}
$$

- the schema $Incorrect$ introduces the function $incorrect$ which takes in the variable $s?$ and returns the variable $s!$
- the variable $s?$ is the component $statement$
- $s!$ is equal to $s?$ if $\$.result.success$ is of the type $INCORRECT$ otherwise $s!$ is an empty set

$$
\begin{array}{|l}
\hline
FilterForIncorrect \\
\hline
\Delta MostDifficultAssessmentQuestions \\
Incorrect \\
incorrects : \mathbb{F} \\
\hline
incorrects \subseteq S_{all} \\
incorrects' = \{s : STATEMENT \mid incorrect(s) \neq \emptyset\} \\
S'_{incorrect} = S_{incorrect} \cup incorrects' \\
\hline
\end{array}
$$

- the set $incorrects$ is a subset of S_{all} which may contain every value within S_{all}
- The set $incorrects'$ contains elements s of type $STATEMENT$ where $incorrect(s)$ is not an empty set
- The updated set $S'_{incorrect}$ is the union of the previous state of $S_{incorrect}$ and $incorrects'$

1.6.5 Processes Results

$$
\begin{array}{|l}
\hline
\textit{GroupByActivityId} \\
\textit{Statements} \\
g? : \mathbb{F} \\
g! : \mathbb{F} \\
group : \mathbb{F} \to \mathbb{F} \\
\hline
g? = statements \Rightarrow \{g : statement\} \\
g! = group(g?) \\
g! = \{groups : \mathrm{seq}_1\ statement \mid \\
\quad \textbf{let}\ \mathrm{seq}_1\ statement_i..statement_j == \mathrm{seq}_1\ s_i..s_j \bullet \\
\quad \forall s_n : s_i..s_j \bullet i \leq n \leq j \bullet s_i.object.id = s_j.object.id = s_n.object.id\} \\
\hline
\end{array}
$$

- The schema *GroupByActivityId* introduces the function *group* which has the input of $g?$ and the output of $g!$
- The input variable $g?$ is the component *statements* which implies its a set of objects g which are each a *statement*
- the output variable $g!$ is a set of objects *groups* which are each a non-empty, finite sequence of *statement* where each member of the sequence $s_i..s_j$ has the same $\$.object.id$

$$
\begin{array}{|l}
\hline
\textit{CountPerGroup} \\
\textit{Statement} \\
c? : \mathrm{seq}_1\ statement \\
c! : \mathbb{N} \\
count : \mathrm{seq}_1\ statement \to \mathbb{N} \\
\hline
c! = count(c?) \\
c! \geq 1 \\
count(c?) = \forall c_n? : \langle c?_i\ ..c?_j \rangle \bullet i \leq n \leq j \land i = 0 \bullet \\
\qquad \exists_1 c! : \mathbb{N} \bullet \textbf{if}\ n = i\ \textbf{then}\ c! = n + 1\ \textbf{else}\ c! = j + 1 \\
\hline
\end{array}
$$

- The schema *CountPerGroup* introduces the function *count* which has the input of $c?$ and the output of $c!$
- The input variable $c?$ is a non-empty, finite sequence in which each element is a *statement*
- The function *count* reads: for all elements $c?_n$ within the sequence $\langle c?_i\ ..c?_j \rangle$, such that n is greater than or equal to i and less than or equal to j, i is equal to zero and there exits a number $c!$ which is equal to $n + 1$ (when $n = i \Rightarrow n = 0$) or equal to n

$$
\begin{array}{|l}
\hline
\textit{AggregateQuestionStatements} \\
\Delta MostDifficultAssessmentQuestions \\
FilterForIncorrect \\
GroupByActivityId \\
CountPerGroup \\
grouped, processed : \mathbb{F} \\
\hline
grouped = \emptyset \\
grouped' = group(S_{incorrect}) \\
S'_{grouped} = S_{grouped} \cup grouped' \\
processed \subseteq S'_{grouped} \\
processed' = \{p : (id, \mathbb{N}) \mid \\
\qquad \textbf{let } \{\langle processed_i \rangle .. \langle processed_j \rangle\} == \{g_i .. g_j\} \bullet \\
\qquad i \leq n \leq j \bullet \forall g_n : g_i .. g_j \bullet \exists\, p_n : p_i .. p_j \bullet \\
\qquad first\, p_n = head\, g_n.object.id \land second\, p_n = count(g_n) \\
S'_{processed} = S_{processed} \cup processed' \\
\hline
\end{array}
$$

- The schema *AggregateQuestionStatements* introduces the variables *grouped* and *processed*
- *grouped* starts as an empty set but then becomes *grouped′* which is the output of applying the function *group* to the set of statements $S_{incorrect}$ created by the opperation *FilterForIncorrect*
- *grouped′* is a set of sequences. The elements of those sequences are statements which all have the same *statement.object.id*
- The set $S_{grouped}$ is updated to the set $S'_{grouped}$ which is the union of $S_{grouped}$ and *grouped′*
- the variable *processed* is a subset of $S'_{grouped}$ which can contain every value within $S'_{grouped}$
- the variable *processed* is updated to be the variable *processed′* which is a set of objects p which are ordered pairs of the component *id* and a natural number. p is defined as:
  - for all sequences $g_i..g_j$ within the set *processed*, there exists an ordered pair p_n such that:
    - the first element of p_n is equal to the *object.id* of the first statement within the sequence g_n.
    - The second element of p_n is equal to the value returned when g_n is passed to the function *count*.
- The set $S'_{processed}$ is the union of the sets $S_{processed}$ and *processed′*

1.6.6 Sequence of Operations

$ProcessedQuestions \mathrel{\widehat=} FilterForIncorrect \fatsemi AggregateQuestionStatements$

- The schema $ProcessedQuestions$ is the sequential composition of operation schemas $FilterForIncorrect$ and $AggregateQuestionStatements$
- $FilterForIncorrect$ happens before $AggregateQuestionStatements$

1.6.7 Return

$ReturnAggregate$

$$\Xi MostDifficultAssessmentQuestions$$
$$ProcessedQuestions$$
$$S_{processed}! : \mathbb{F}$$

$$S_{processed}! = S_{processed}$$

- The returned variable $S_{processed}!$ is equal to the current state of variable $S_{processed}$ after the operations $FilterForIncorrect$ and $AggregateQuestionStatements$

1.7 Pseudocode

Algorithm 1: Most Difficult Assessment Questions

Input: S_{all}, $displayN$
Result: $display''$
$context = \{\}$;
$display = []$;
while $S_{all} \neq \emptyset$ **do**
  foreach $s \in S_{all}$ **do**
    if $s.result.success = INCORRECT$ **then**
      do
      $S'_{incorrect} \leftarrow s \cup S_{incorrect}$;
      $S'_{all} \leftarrow S_{all} \setminus s$;
      recur $S'_{all}, S'_{incorrect}$;
    else
      do
      $S'_{all} \leftarrow S_{all} \setminus s$;
      recur S'_{all}
    end
  end
end
while $S'_{incorrect} \neq \emptyset$ **do**
  foreach $si \in S'_{incorrect}$ **do**
    $id \leftarrow si.object.id$;
    if $id \notin context$ **then**
      do
      $count = 1$;
      $context' \leftarrow \{id : count\}$;
      $S'_{incorrect} \leftarrow S_{incorrect} \setminus si$;
      recur $context', S'_{incorrect}$;
    else
      do
      $count' \leftarrow inc(context.id)$;
      $context' \leftarrow \{id : count'\}$;
      $S'_{incorrect} \leftarrow S_{incorrect} \setminus si$;
      recur $context', S'_{incorrect}$;
    end
  end
end
foreach $id \in context'$ **do**
  $IdToCount \leftarrow [id, context.id]$;
  $display' \leftarrow display \frown IdToCount$;
  recur $display'$
end
return $display'' \leftarrow$ **take(sortBySubArray(**$display'$**),** $displayN$**)**

- The Z schemas are used within this pseudocode
- The return value display is an array of length display-n, where each element of display is an array of [*statement.object.id*, #] where # representing the number of times *statement.object.id* appeared within $S'_{incorrect}$

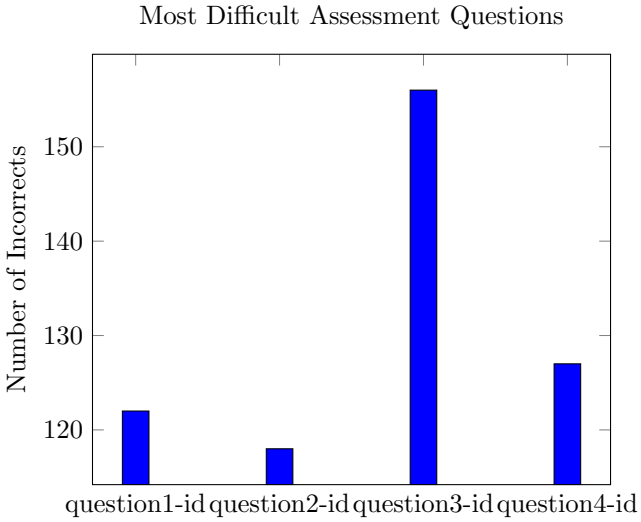
1.8 JSON Schema

```
{"type":"array",
  "items":{"type":"array",
    "items":[{"type":"string"}, {"type":"number"}]}}
```

1.9 Visualization Description

The **Most Difficult Assessment Questions** visualization will be a bar chart where the domain consists of *statement.object.id* and the range is a number greater than or equal to 1. Every subarray within the array display will be a grouping within the bar chart. The pseudocode specifies an input paramter display-n which controls the length of the array display and therefor the number of groups contained within the visualization.

1.10 Visualization prototype



1.11 Prototype Improvement Suggestions

Additional features may be implemented on top of this base specification but they would require adding aditional values to each subarray returned by the algorithm. These additional values can be retrieved via (1) performing metadata lookup within or independently of the algorithm (2) by utilizing additional xAPI

statement paramters and/or (3) by performing additional computations. The following examples assume the metadata is contained within each statement available to the algorithm.

- Use the name of the activity for the x-axis label instead of its id.
  - $\$.object.definition.name$
  - grouping of statements should still happen by $\$.object.id$ to ensure an accurate count
- a tooltip containing contextual information about the question such as:
  - The question text
    - $\$.object.definition.description$
  - Interaction Type
    - $\$.object.definition$ which contains interaction properties
  - Answer choices
    - $\$.object.definition$ which contains interaction properties
  - Correct answer
    - $\$.object.definition$ which contains interaction properties
  - Most popular incorrect answer
    - This would require an extra step of processing and all statements would need to utilize interaction properties within $\$.object.definition$
  - average partial credit earned (if applicable)
    - $\$.result.score.scaled$
    - The one potential issue with using scaled score is the calculation of scaled is not stricly defined by the xAPI specification but is instead up to the authors of the LRP. This results in the inability to reliably compare scaled scores across LRPs.
    - if $\$.result.score.raw$, $\$.result.score.min$ and $\$.result.score.max$ are reported for all questions, it becomes possible to reliably compare scores across questions and LRPs.
  - average number of re-attempts
    - this would require additional steps of processing so that $\$.actor$ is considered as well
    - due to the problem of actor unification, ie the same person being identified differently across statements, this metric may not be accurate.
  - average time spent on the question
    - $\$.result.duration$
    - this would require additional steps of processing to extract the duration and average it.

- a tooltip containing contextual information about the course and/or assessment the question was within
  - the instructor for the course
    - $.*context.instructor*
  - competency associated with the question and/or course
    - $.*context.contextActivities*
  - metadata about the learning content associated with the question such as average time spent engaging with associated content before attempting the question.
  - this would require additional steps of processing to retrieve metadata about the content and its usage.
    - $.*context.contextActivities*