Zeeguu Internship - Final Report

Timon Back (s3218147) & Peter Ullrich (s2273942)

Block 2B - 2016/17

1 Introduction

In the first weeks of our Internship, we researched promising scheduling algorithms, which could help to optimize the learning rate of users of the Zeeguu system. Zeeguu is a research project by Prof. Mircea Lungu and aims to make learning languages quick and fun by enabling learners to read texts written in foreign languages and save and exercise any unknown words found in these texts. We focused on and discussed two algorithms with Prof. Lungu of the University of Groningen, which were the **A**daptive **R**esponse **T**ime based **S**equencing (ARTS) algorithm by Mettler, Massey, and Kellman (2011) and the DASH algorithm proposed by Lindsey, Shroyer, Pashler, and Mozer (2013)[3], which focuses on item **D**ifficulty, the students' **A**bility, and the individual **S**tudy **H**istory. The ARTS and DASH algorithms both aim to create a schedule, which maximizes the positive impact of the spacing effect[1] on learning.

The DASH algorithm was implemented and tested with 179 third-semester Spanish learning students, which was according to Lindsey, Shroyer, Pashler, and Mozer (2013) the biggest sample group of any research on scheduling algorithms so far. The DASH algorithm is based on a latent-state Bayesian model, which gives it convincing statistical credibility, however its root in intricate Statistics made it hard to grasp and adjust to our needs. In their experiment with Spanish as foreign language learners, the authors showed that participants were more likely to correctly recall learned words in an exam 28 days after the Spanish course ended when they learned the words according to a schedule created using the DASH algorithm. The ARTS algorithm on the other hand was not heavily mathematical, but very straight forward and easy to understand, which also meant that we could adjust it according to our needs. The authors Mettler, Massey, and Kellman tested the ARTS algorithm on undergraduate students, who had to learn and correctly recall the geographical location of 24 African countries. The authors conducted this experiment two times, once in 2011 with 50 participants and once in 2016 with 72 participants. The results of these experiments showed that the participants were better at recalling the location of African countries when they learned them according to an adaptive schedule created and updated by the ARTS algorithm after each exercise.

Although the DASH algorithm had a more convincing mathematical foundation, we decided on using the ARTS algorithm instead since it offered more possibilities to adapt the algorithm to the Zeeguu context.

2 The ARTS Algorithm

The ARTS algorithm as proposed by [3] is based on the following formula in Equation 1. The combined priorities of all words that a user should learn defines an order in which the words are repeated. The word with the highest priority gets repeated next.

$$Priority = a * (N - D) * [(1 - \alpha) * b * log(\frac{RT}{r}) + \alpha * w] \tag{1}$$

where the factors stand for:

a	A global linear scaling factor
N	Number of iterations that have passed since this word has been practised last
D	Penalty for words that have been practised shortly before[1]
α	Indicates with 0 (correct) and 1 (incorrect) the outcome from the last practise
b	Scaling factor for words that have been answered correctly the last time
RT	The reaction time for giving the answer[2]
r	Scaling factor for the reaction time (inverted)
w	Scaling factor for words that have been answered incorrectly the last time

As one notices, the priority of an item only depends on the last practise result. The complete history for that item is not taken into consideration. This makes the algorithm light-weight during calculation. Still, the algorithm needs to be re-run for every learning iteration as indirectly the time of last practice for each word $(N - D)$ is part of the formula.

[1]We measure this in a simplified learning sessions. Every word creates a new session.
[2]The unit does not matter, as long as always the same unit is used. We use milliseconds.

3 Implementing the ARTS Algorithm in Zeeguu Core

We decided on implementing the ARTS algorithm in the Zeeguu Core[3] code base, which functions as connection between the database and the Zeeguu API, which in turn is used by the Zeeguu App, the front-end component of the Zeeguu system. The whole Zeeguu system resolves around the `Bookmark` class, which represents a word that a user wants to learn. Whenever a `Bookmark` is learned, an `Exercise` object is added to its `exercise log`. The `exercise` instance holds, among others, information about whether the `Bookmark` was recalled correctly and the time it took until the `Bookmark` was recalled. Thus, the `Exercise` class holds all information needed for the ARTS algorithm.

3.1 Algorithm

The implementation of the ARTS algorithm is done as shown below. It takes the formula proposed by [3] and also uses their default values for the parameters.

A single instance of the algorithm does not depend on any other class. The `calculate` method can be called as often as needed to calculate the priority of any bookmark.

```python
class ArtsRT:
    """
    ARTS algorithm with default values as described in:
    Adaptive response-time-based category sequencing in perceptual learning
    by Everett Mettler and Philip J. Kellman

    a: Constant - general weight
    D: Constant - enforced delay (trials)
    b: Constant - weight for the response time
    r: Constant - weight for the response time (inside log)
    w: Constant - priority increment for an error. Higher values let
        incorrect items appear quicker again
    """

    def __init__(self, a=0.1, d=2, b=1.1, r=1.7, w=20):
        self.a = a
        self.d = d
        self.b = b
        self.r = r
        self.w = w

    def calculate(self, args):
        """ Calculate the ARTS priority
        Parameters:
            args: Contains the following parameters:
                N: number of trials since item was presented
                alpha: 0, if item was last answered correct; 1 otherwise
                RT: response time on most recent presentation
        """
        N, alpha, RT = args
        return self.a \
```

[3]`https://github.com/mircealungu/Zeeguu-Core`

```
                * (N - self.d) \
                * ((1 - alpha) * self.b * math.log(RT / self.r)
                    + (alpha * self.w))
```

3.2 AlgorithmService

The AlgorithmService is a controller that handles the calculation of bookmark priorities based on the ARTS algorithm. Its main function is the `update_bookmark_priorities()` function, which updates and stores the priorities for all bookmarks of a give user. The algorithm instance, which ultimately calculates the priority for a given bookmark using its last exercise, is retrieved from the ABTesting class, which handles the distribution of algorithms over the bookmarks so that each sample group has the same size.

3.2.1 A/B Testing

In order to test different sets of parameters for the ARTS algorithm, we implemented an ABTesting class, which functions as a controller for picking an algorithm instance to calculate the priority of a Bookmark. The class holds a list of algorithm instances, which are loaded from the `algorithms.ini` file. Any combination of parameters that were chosen for implementation can be defined in the `algorithms.ini` file. Algorithms can be of any type that is specified as an algorithm class in the `zeeguu.algos.arts` package. At the moment, the supported algorithm types are the standard ARTS algorithm using the reaction time (class ArtsRT), a random algorithm (class ArtsRandom), that returns a random priority, and two types (ArtsDiffSlow and ArtsDiffFast) which use the standard deviation of the reaction time to the population mean of an exercise instead of the reaction time alone. More about these last two in subsection 3.3.

Whenever the AlgoService is asked to update the priority of a bookmark, it will ask the ABTesting class for an algorithm with which the priority will be calculated. The ABTesting takes the modulo of the bookmark ID with the number of algorithms available and returns the algorithm with the index equal to the result of this calculation. By using this method, we ensure equally sized sample sizes for each of the algorithms that are tested. However, an algorithm can be chosen based on any Integer that is passed to the `get_algorithm_for_id()` function in ABTesting.

3.3 ARTS with Standard Deviation

One of the advantages of the Zeeguu system is the variety in different exercises types like word recall[4], word matching[5] and word identification[6]. Every word is learned in multiple exercise types as those exercise types have different difficulties levels. So, first a word just needs to be identified in a context (easy) and later recalled directly (difficult).

The ARTS algorithm uses besides the correctness of the last answer also the response time. As one can imagine, the response time will be different between the different exercise types. To overcome the differences in response time throughout different exercise types and to make the data more comparable, we use the standard deviation to normalize the response time:

$$Priority = a * (N - D) * [(1 - \alpha) * b * e^{r*sd} + \alpha * w] \tag{2}$$

In comparison to the original Equation 1, $log(\frac{RT}{r})$ was replaced in Equation 2 by e^{r*sd}. However, sd is not exactly the response time. Instead, sd is the difference of the reaction time (rt) from the $mean$ of an exercise, divided by the standard deviation std of all reaction times for that exercise type. So: $sd = \frac{rt-mean}{std}$. In order to have the $mean$ and the std value available, those are computed on the existing data in `AlgoService.update_exercise_source_stats()` for each exercise type. Those should be re-computed every now and then - when more completed exercises are available. So, this essentially behaves like a long-term cache.

Due to the fact of a power-function e^{r*sd}, the algorithm with standard deviation weights differences between slow (high) reaction times more than between fast (low) reaction times (a higher sd results in a much bigger e^{r*sd}). However, desired would be that fast reaction times have an higher impact than slower ones - if answering takes very long, then more practise is definitely required. Therefore, the formula is adapted to:

$$Priority = a * (N - D) * [(1 - \alpha) * \frac{b}{e^{r*sd}} + \alpha * w] \tag{3}$$

Here, we divide the constant b by the e^{r*sd} factor, which yields bigger values (i.e. priorities) when reaction times are faster than the population mean (thus sd is negative), than when the reaction times are slower than the mean (thus sd is positive). Which and whether at all a version of the algorithms with standard deviation is used eventually was not decided during the internship and is thus entirely up to Prof. Lungu.

[4]The user has to enter the word in target language based on the word in the native language.
[5]The user has to select the correct translation out of multiple options.
[6]The user has to identify the word in a sentence in the target language.

Equation 1 does not differentiate between different exercise types, but Equation 2 and Equation 3 do. All of those are implemented in code[7]. We recommend Equation 3 as it puts an higher importance on fast responses.

3.4 Unit Testing

After we implemented the code for the ARTS algorithm, we planned on writing unit tests to test its functionality and robustness. However, we ran into problems while integrating our unit tests into the existing testing environment. The Zeeguu unit tests all relied heavily on a script that populated the test database with (non-random) data and tested the Zeeguu functionality against this data. In order to test our algorithm implementation, we had to add additional data to the populate script, but this appeared to break other unit tests, which was obviously not our intention. After eventually integrating our unit tests into the existing testing framework, we decided to refactor the testing environment with certain guidelines in mind:

1. Test Data should be truly random
2. Unit tests can add, edit,and delete the test data without consequences
3. Unit tests have to be truly independent of each other

After discussing our plans with Prof. Lungu, we started refactoring the testing framework. We first implemented Rules (more about this in subsubsection 3.4.1), which are a common testing method in Java development[8]. These rules allowed us to add random testing data to the test database whenever an unit test needed it. For creating the random test data we used the Faker[9] framework, which offers a wide range of random data types (e.g. URLs, names, emails, words, sentences, etc.).

After we implemented the new test data creation framework, we changed the location of the test database from a mysql database to an in-memory sqlite database. This sped up the execution time of our tests from around 3 minutes to just 20 seconds.

3.4.1 Rules

We used the testing rules concept to populate the testing database with random data on which we ran the tests. The JUnit documentation defines rules in the following way:

[7]Equation 1 is in `arts_rt.py`, Equation 2 in `arts_diff_fast.py` and Equation 3 in `arts_diff_slow.py`

[8]`http://www.vogella.com/tutorials/JUnit/article.html#junitadvanced_rules`

[9]`https://github.com/joke2k/faker`

Rules allow very flexible addition or redefinition of the behavior of each test method in a test class.[10]

In our case, we created rules for any object (e.g. Bookmark), which me might have to add to the database before a test can be run. For creating bookmarks, for example, we created a `BookmarkRule`, which takes a user object as argument and creates a random bookmark for the given user. Noteworthy here is that instances of all classes on which the `Bookmark` object relies (e.g. `Language`, `Url`, `ExerciseOutcome`, etc.) are created as well when a `Bookmark` is created using the `BookmarkRule`.

3.5 Determine the algorithm parameters

Determining the parameters turned out to be a more difficult task than expected in the beginning. In essence, the parameters depend on the objectives of the learning. For now, it was decided that the parameters should be adjusted in a way, that 20 words are being practiced in parallel.

In order to derive the parameters for either of the Equation 1, Equation 2 and Equation 3, the following constants need to be found: b, D and w. As those depend on each other, an iterative approximation approach is chosen. The code can be found in the file `algo_parameter_approximator.py`. If the found improvement gets too small or after the 30th iteration no improvements are found, the approximation finishes.

The model calculates in each iteration the priority of all words that are currently being studied and 'learns' the word (adds a new random exercise to the exercise log) with the highest priority, thus giving it a lower priority in the next iteration. In the beginning all words are given a constant, positive `MAX_PRIORITY` score. Words are considered as learned as soon as they have been answered three times in a row correctly and are not repeated any further. This metric seems reasonable, but is subject to change - especially since the Zeeguu system uses user feedback at the moment. In the future, with the use of machine learning even the recall probability can be determined.

To also account for various word difficulties, each word is assigned a likelihood-to-be-answered-correct value, which makes it more difficult to learn certain words. For this, the model uses the existing data to incorporate the data about the reaction time as well as how often a way has been answered correctly. So, the algorithm should be run again every now and then to update the parameters.

[10]`https://github.com/junit-team/junit4/wiki/Rules`

4 Personalizing / Individualizing the Parameters of ARTS for every Learner

To verify the ARTS algorithm, we have it run on all users that are using the Zeeguu system. As of 16.06.17, there are in total 666 users, from which 255 have used the system to also create a learning exercise (bookmark). The calculation took 195 seconds. With our parameter-fitting script in `algo_parameter_approximator.py`, we calculated values for b, D, and w so that the results from the ARTS algorithm best fit a given 'Hyper-parameter'[11], which in our case was how many words a user learns simultaneously. We let our script calculate the b, D, and w parameters, so that the users would learn only 20 words simultaneously. In Table 1 the summarized results and in Figure 1 the detailed results are presented.

Some pre-processing was done on the data beforehand to ignore invalid entries. From the 2017-06-12 database dump all exercise entries that were created before 2017-05-02 and all exercises, which reaction time (solving_speed) is negative are excluded[12].

			range		
Parameter	**mean**	**median**	**from**	**to**	**Literature**
b	1.28	1.10	0.15	21.10	1.1
D	7.13	3.25	0.12	27.00	2
w	24.57	22.50	5.00	50.00	20

Table 1: Summarized overview of the calculated parameter values b, D, w with the ARTS algorithm for each user according to the database dump from 2017-06-12

First, the summarized (median) values in Table 1 are quite close to the original values proposed by the authors of the ARTS algorithm. Although the starting point were the original values, no extraordinary shift towards one extreme is seen - except for the d parameter. But this was expected as due to the chosen goal of 20 concurrent words for which D is parameter with the highest influence[13].

[11] `https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)`

[12] See the appendix for script-based import and filtering

[13] When the goal is to have 10 concurrent words, the median of D is 5.19 (mean: 2.00)

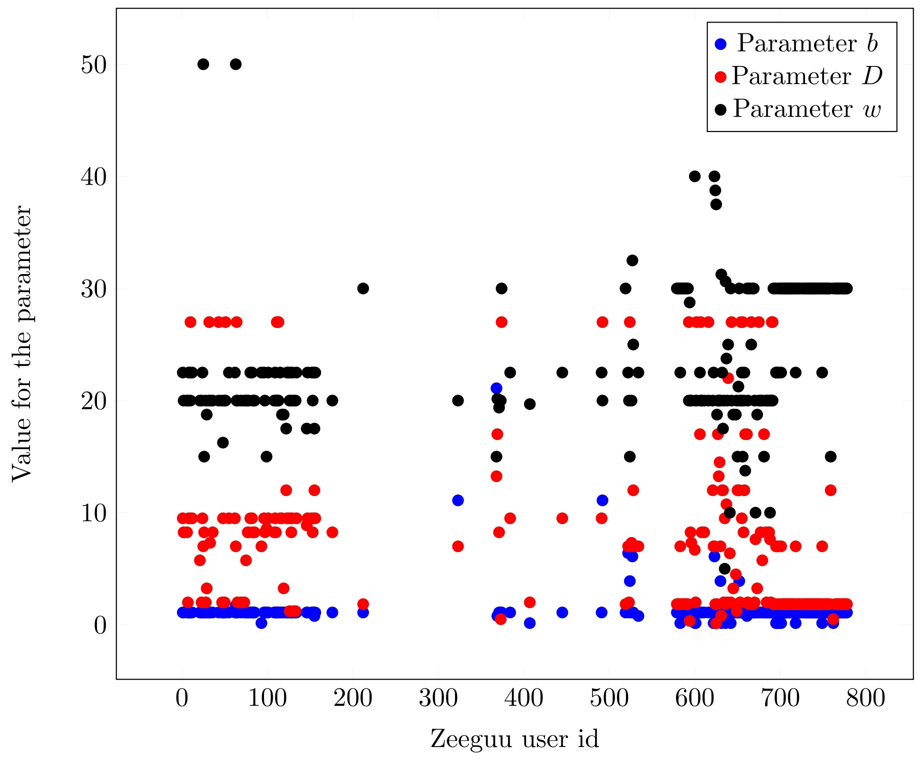

Figure 1: Detailed overview of the calculated parameter values b, D, w with the ARTS algorithm for each user according to the database dump from 2017-06-12

In Figure 1 the detailed results are presented. For each user id exist three dots corresponding to the parameters b, D, w. As some users (x-axis) are not active, some gaps are present especially in the range of 200 to 350 or even till 600.

In comparison to the first users (low user ids), users that joined later have a lower parameter for D and higher w parameter. This is due to the fact that newer users have not studied so many words, neither added so many words to their account. Thus, all words need to be repeated more often.

Also, the graph shows in the user id range starting from 600 clustered dots with the same value for multiple users. This is also due to new users which have not such a diverse data set yet. Still, those values can be used as default values for new users.

This shifts the focus for the analysis towards the left hand side of the graph. The parameters cluster around the mean values and differ to some extend from the median values. Therefore, the parameters should be adjusted over time to reflect each users own learning history.

4.1 Algorithm with Standard deviation

We also applied the improved algorithm with standard deviation from Equation 3. In fact, different mean values exist between the exercise types, proving the assumption that each exercise types has a different reaction speed.

			range		
Parameter	**mean**	**median**	**from**	**to**	**Literature**
b	1.41	1.10	0.15	37.00	1.1
D	10.33	7.00	0.50	37.00	2
w	19.16	20.00	0.00	30.00	20

Table 2: Summarized overview of the calculated parameter values b, D, w with the ARTS algorithm *with standard deviation as proposed in Equation 3* for each user according to the database dump from 2017-06-12

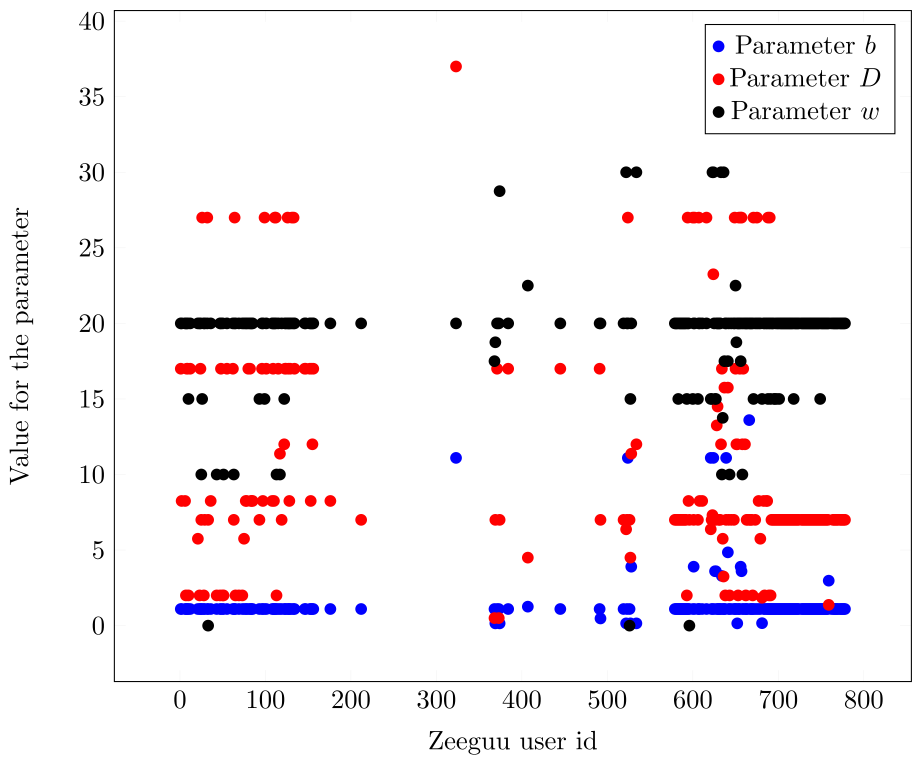

Figure 2: Detailed overview of the calculated parameter values b, D, w with the ARTS algorithm *with standard deviation as proposed in Equation 3* for each user according to the database dump from 2017-06-12

One remark needs to be made for Figure 2. In the user id range of 700 to 800, a sudden jump of the w parameter to 30 is recognizable compared to the original ARTS algorithm in Figure 1. The reason is that those Zeeguu user have no complete exercises yet and so default data is assumed, which produces different results. A closer look at the data is taken in subsection 4.2.

4.2 Diving deeper into the data

In Figure 1 straight horizontal lines can be identified, as well as missing data points and dots spread out over the complete y range. Therefore, we had a look at the database dump and how many bookmarks and exercises each Zeeguu user has created. Although the data about the amount of exercises per user is relevant, for further research, we also included data about the amount of bookmarks[14].

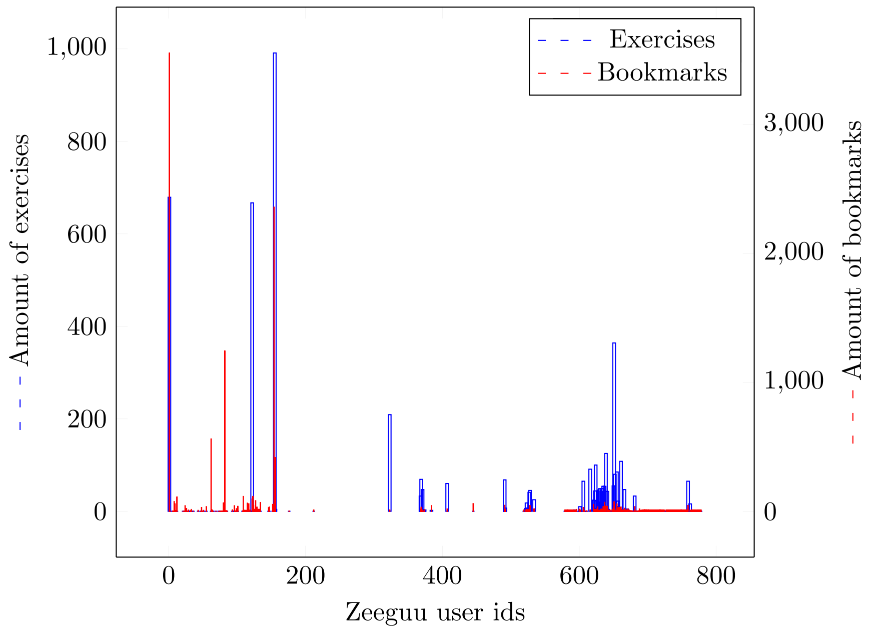

Figure 3: Amount of exercises and bookmarks for each user according to the database dump from 2017-06-12

With the visualization of Figure 3, we derived the following statements:

- Whenever no parameters could be calculated for a Zeeguu user, then also no exercise or bookmark exists for that user (see especially user ids 200

[14]Every exercise is always linked to a bookmark. One exercise has always one bookmark, one bookmark can have multiple exercises (One-To-Many)

to 300).

- The more exercises one Zeeguu user completes, the more likely and further do the individual algorithm parameters move away from the default values.
- The ratio between exercises and bookmarks has no effect on the parameters for the algorithm.
- When people start using the Zeeguu system, they usually have more completed exercises than bookmarks. So, users actually use the system to repeat words to learn them and not only bookmark them.

5 Conclusion

The main focus of our internship was implementing the ARTS algorithm, but this was done surprisingly quickly, which gave us the opportunity to contribute to the Zeeguu project as a whole. We decided on refactoring the testing framework, which was needed for our unit tests, but also improved the quality of other unit tests, which were always testing against a non-random dataset. After refactoring the testing framework, we turned our attention back to the ARTS algorithm and tried to extrapolate its performance based on real data. We were surprised again how easy it was to tweak the algorithm based on a given requirement like e.g. a user should learn only 20 words simultaneously.

In conclusion, we were content with our decision to use the ARTS algorithm instead of the DASH algorithm since our assumption that the ARTS algorithm would be better customizable than the DASH algorithm was vindicated. We think that we implemented the ARTS algorithm successfully and improved the overall quality of the Zeeguu code. We further hope that the algorithm will be useful in the future as well.

References

[1] Harry P Bahrick, Lorraine E Bahrick, Audrey S Bahrick, and Phyllis E Bahrick. Maintenance of foreign language vocabulary and the spacing effect. *Psychological Science*, 4(5):316–321, 1993.

[2] Mohammad M Khajah, Robert V Lindsey, and Michael C Mozer. Maximizing students' retention via spaced review: Practical guidance from computational models of memory. *Topics in cognitive science*, 6(1):157–169, 2014.

[3] Everett Mettler and Philip J Kellman. Adaptive response-time-based category sequencing in perceptual learning. *Vision research*, 99:111–123, 2014.

A Files used to import and filter the database dump

A.1 Import a database dump

```
#!/bin/bash

USER=root
PWD=root
DATABASE=zeeguu_live

echo "Reset database"
echo "DROP DATABASE $DATABASE;" | mysql -u $USER -p$PWD
echo "CREATE DATABASE $DATABASE;" | mysql -u $USER -p$PWD

CMD="mysql -u $USER -p$PWD $DATABASE"

echo "Import database dump"
$CMD < zeeguu_2017-06-12.sql
```

A.2 Filter the database dump

```
#!/bin/bash

USER=root
PWD=root
DATABASE=zeeguu_live

CMD="mysql -u $USER -p$PWD $DATABASE"

# remove the foreign key to allow deleting later
echo "ALTER TABLE bookmark_exercise_mapping DROP FOREIGN KEY
    bookmark_exercise_mapping_ibfk_1" | $CMD

echo "DELETE FROM exercise WHERE solving_speed < 0 OR time < '2017-05-02'" |
    $CMD
```