Optimizing teaching sequences using Multi-Armed Bandits

François Darmon
francois.darmon@ens-paris-saclay.fr

Yousri Sellami
yousri.sellami@ens-paris-saclay.fr

Abstract

In this project, we approach Intelligent Tutoring Systems (ITS) using the Multi-Armed Bandits framework. Inspired by the work of [3], we model the interactions between the teacher and a class of students as a reinforcement learning problem. We propose an algorithm that maximizes the progress of each student in each competence given time and motivation resources. This algorithm relies on an online estimation of students competences to propose the right activity at the right time. The system is evaluated in a scenario where a 7-8 year old schoolchildren learn how to decompose numbers while manipulating money. We run simulations on different types of students and show that our algorithm outperforms several baselines.

1 Introduction

A teacher wants his students to acquire a set of skills (summing, subtracting and multiplying numbers for example). He can propose a sequence of different activities to his class (multiple choice questions, concrete exercises with items, videos...). The goal of Intelligent Tutoring systems is to find the teaching sequence that maximizes the average competence level over all skills. This is a challenging question first because obviously, students are different : they don't have the same level at the beginning of the experiment and they don't learn at the same pace. Second, the teacher has limited time meaning that he can only propose a limited number of activities. Third, he needs to keep students motivated by not proposing too difficult or too easy exercises.

Reinforcement Learning is frequently used for ITS. One famous approach is the partial-observable Markov decision process (POMDP) [4]. This approach uses a POMDP modelisation of the students. This modelisation is similar to Markov decision process except the states are not observable. Instead, observations are provided throughout the experiment. The states are the level of understanding of the student and the observations are the ways the student interacts with the ITS. Although the transition probability can be learnt with a sufficient number of samples, there need to be a modelisation step that defines the states and the observations. Such definitions are strong modelisation of the students that may not be realistic.

Our modelisation uses the Multi-Armed-Bandits framework that does not rely on strong assumptions. Instead of assuming a POMDP structure of the student learning, we only assume that there exist separate Knowledge Competences (KC) that need to be acquired. We also assume that the level of understanding of each KC can be encoded using a real number.

First, we present the teaching and student model and the MAB setting. Second, we introduce our algorithm based on Exp3, which proposes the right activity at the right time by estimating the competences of each student. Finally, we compare the performances of our algorithm to several baselines through different experiments.

2 Modelisation

This section describes the modelisation used for modelling exercises, student and their interaction. A concrete example of this modelisation will be explained in section 4.

2.1 Teaching scenario

We model the problem that students are asked to solve by a set of n_c competences to master so that each student has n_c knowledge competence (KC). The goal of the intelligent tutoring system is to provide to student the best sequence of exercises in order to maximize each KC. The ITS can only interact with the student by proposing an exercise and seeing whether the answer is correct or not.

The ITS knows beforehand the function q such as $q(a)$ is the estimated required values in each KC for a student to succeed in activity a. This length n_c vector is a way to model the difficulty of each exercise. It is supposed that a student with every KC below $q(a)$ will be less likely to succeed in activity a than a student with KC above $q(a)$.

Each exercise is parametrized with n_p parameters, that can take $n^a_1, \dots, n^a_{n_p}$ values. An exercise (or activity) is then a length n_p vector $(a_1 \dots a_{n_p})$. In order to simplify the management of function q, the difficulty of an activity can be factorized as follows :

$$q(a_1, \dots, a_{n_p}) = \prod_{i=1}^{n_p} q_i(a_i) \tag{1}$$

Such factorization makes the assumption that every component of an activity contributes independently to the overall difficulty of the activity. This assumption may seem unrealistic but it is a way to harness the combinatorial explosion of the number of possible activities for each combination of parameters.

2.2 Intelligent Tutoring Systems using Multi-Armed-Bandits

2.2.1 The Multi-Armed Bandit setting

We model our problem using the MAB framework. Following a casino analogy, multi-armed bandits describe the problem of identifying the machine that provides the maximum reward, initially unknown. To do so, the agent needs to spend money exploring all of them before being able to bet always on the best one: this is called the exploration/exploitation trade-off. In our case, the gambler is replaced by the ITS, the choice of machines is replaced by a choice of a learning activity, and reward is replaced by learning progress of the student.

At each round, the ITS pulls an arm (chooses an exercise) and gets a reward which measures the learning progress (depending on whether the student has succeeded or not). In fact, following factorization (1), at each round choosing an activity $a = (a_1, \dots, a_p)$ is equivalent to pulling n_p simultaneous MAB: one for each activity parameter, each with n^a_i arms (number of values for each activity parameters, see Figure 1).

This MAB problem is not easy because the reward function is clearly **non-stationary**. Indeed, easy exercises should stop making the student progress once he has reached a certain level of competence. An arm that corresponds to an easy exercise would give a high reward at the beginning when it is the most relevant exercise for the student to make progress. But after several exercises, this exercise becomes less and less useful and its reward decreases. Similarly, an arm associated to a difficult exercise will have its associated reward increase as the experience goes.

2.2.2 Reward function

The reward function cannot be as simple as success or failure at each round because the best policy would be to select at each time the easiest activity. The reward function must quantify the **progress** made by the student with the exercise proposed.

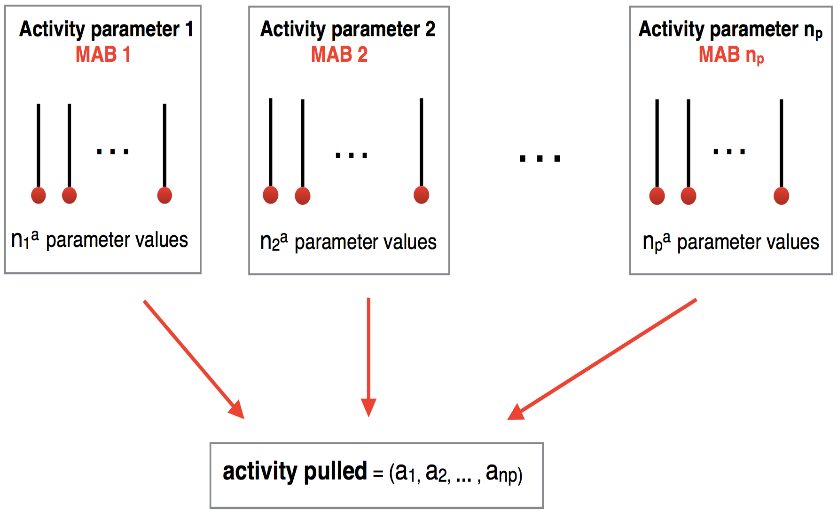


Figure 1: MAB architecture

In the modelisation we used, the ITS keeps an estimation of the KC of the student $\widehat{KC}$ and combines it with the difficulty of the exercise $q(a)$ to compute the reward $r(a)$.

$$r(a) = \begin{cases} \sum_{i=1}^{n_c} \max(q(a)_i - \widehat{KC}_i, 0) & \text{if success} \\ \sum_{i=1}^{n_c} \min(q(a)_i - \widehat{KC}_i, 0) & \text{if failure} \end{cases} \tag{2}$$

In other words, the student can make progress only if he answers correctly to an exercise with a difficulty higher than his estimated competence. On the contrary, he gets a negative reward if he fails to answer an exercise easier than his estimated competence.

One should note that the reward function as defined above uses the estimated student competence to quantify the learning progress instead of the true competence which is unknown. Therefore, the reward is a good approximation of the true learning progress only if we manage to estimate accurately students competence at each round.

Since the reward function depends on our competence estimation, arms have no fixed distributions meaning that we are facing a **non-stochastic** multi-armed bandit problem.

2.2.3 Definition of regret

The regret can be defined as the difference between the expectation of reward for the best possible combination of arms and the expectation of the reward for the chosen combination of arms. The cumulative regret can then be defined as

$$R_t = \sum_{i=1}^{t} \left(\max_a \mathbb{E}_i \left[r(a) \right] - \mathbb{E}_i [r(a(i))] \right) \tag{3}$$

Where $a(i)$ is the activity chosen at time i and $r(a)$ is the reward computed with (2) using the true value of each KC. The cumulative regret can be computed when the probability of success is known for every activities and when the true level in each KC are known.

2.2.4 Specific constraints of the MAB

The first constraint is that we should limit exploration in order not to lose student motivation. Indeed, providing too easy or too difficult exercises is in both case a way of demotivating students.

The second constraint we introduce is that some activities cannot be proposed if the student estimated competences are below certain values. In other words, as the student learns progressively, the number of arms grows meaning that more and more activities can be selected by the algorithm. These pedagogical restrictions were designed by an expert. The set of possible activities at a given time is called Zone of Proximal Development (ZPD). For example the expert knows that for most students it will be useless to propose exercises about decomposition of real numbers if they do not know how to add simple integers It enables to improve students motivations and facilitate exploration by excluding exercises that are way too difficult.

2.3 Student Model

For testing purpose, we need to have a modelisation of students. This modelisation will not be used by the algorithm and any other modelisation might be useful for the test. It needs two features: a probability of success for an activity and a KC evolution scheme. We used the following modelisation for the probability of success:

$$p(a) = \prod_{i=1}^{n_c} \left(\frac{\lambda(a)}{1 + \exp\left(\alpha + \beta(KC_i - q(a)_i)\right)} \right)^{1/n_c} \tag{4}$$

where α and β are tunable parameters : α corresponds to the probability of success when student competences fit the required competences by the exercise i.e $\forall i, KC_i = q(a)_i$ and β corresponds to how fast the probability of success drops or rises when $KC_i - q(a)_i$ drops or rises.

The function λ is used to define two different kinds of students. The first kind is called Q-Students. For these students, $\lambda(a) = 1$ for all a which means that they can succeed in any activities provided they have the prerequisite knowledge. The second kind is called P-Students. They can succeed in most of the activities but there are some activities they cannot solve even if they have perfect knowledge of each KC. For these activities, $\lambda(a) = 0$. This separation between P and Q students is useful to modelize students who have comprehension blockages. For example, a student who cannot read will not be able to solve any written exercises.

The Knowledge Competence evolution scheme is simple, a student makes progress in a specific KC when the exercise is a success and his level in the KC is lower than the difficulty of the exercise. KC_i is updated by $KC_i = \eta(q_i(a) - KC_i)$ where η is the student learning rate.

Once students' probability of success is modelled, it is possible to compute $\max_a \mathbb{E}_i\left[r(a)\right]$ and $\mathbb{E}_i[r(a(i)]$ in equation (3) and use it to compute the cumulative regret.

$$\begin{aligned}\mathbb{E}_i[r(a(i)] = &\sum_{j=1}^{n_c} p(a(i)) \cdot \max(0, q(a(i))_j - KC_j(i)) + \\ &\sum_{j=1}^{n_c} (1 - p(a(i))) \cdot \min(0, q(a(i))_j - KC_j(i))\end{aligned}$$

3 Algorithm

3.1 Baselines

We implemented two baselines to compare our results with :

- **Random**: a random selection of activity parameters among possible values given by the ZPD.
- **Expert predefined sequence**: a sequence of activities of increasing level. One student starts with the first activity then moves to the next one after succeeding a fixed number of times that kind of activity. This sequence needs to be designed beforehand by an expert who knows the relative complexity of the exercises. As the number of parameters increases, it quickly becomes impossible for the expert to add every possible combination of parameters in the sequence.

3.2 Exp3

Although [3] uses a variant of Exp4 algorithm for the ITS, we have chosen to implement a simpler version Exp3 for dealing with the adversarial bandit problem [1]. Its pseudo-code can be found in algorithm 1.

Exp3 explores uniformly with probability γ, and draws an activity according to the recent rewards of each activity with probability $1 - \gamma$.

In the adversarial setting, we only observe the reward of the activity chosen. To deal with this lack of information, Exp3 uses the **importance trick**. It consists in associating null rewards to non-observed arms and estimating the observed reward by dividing it by the probability of the pulled arm. Intuitively, if an activity gets important rewards at the beginning and then becomes irrelevant, we would like the algorithm to propose a new activity. The problem is that because this new activity has just started to become relevant, it has not received good rewards yet and thus has low probability of being selected. The importance trick compensates for this low probability by inflating the observed reward. In the next rounds, it will make this new relevant activity more likely to be selected.

From a mathematical point of view, this inflation of the observed reward is a good idea because the estimated reward is an unbiaised estimator of the expected reward for each arm:

$$\mathbb{E}(\widehat{r_{i,a_i}}) = p_{i,a_i} \cdot \frac{r_{i,a_i}}{p_{i,a_i}} + (1 - p_{i,a_i}) \cdot 0 = r_{i,a_i}$$

Algorithm 1: Exp3 algorithm

```
/* Initialization                                                      */
for $i = 1 \cdots n_p$ do
    $w_i \leftarrow (1/n_i^a \cdots 1/n_i^a)$
end
$c_L \leftarrow$ initialization of competence level
while learning do
    for $i = 1 \cdots n_p$ do
        $p_i \leftarrow (1-\gamma)w_i + \gamma \cdot (1/n_a^i \cdots 1/n_a^i)$
        $a_i \leftarrow$ Sample parameter proportional to $p_i$
    end
    $success \leftarrow$ Propose activity $a = (a_1 \cdots a_{n_p})$
    /* Compute reward                                                  */
    if success then
        $r \leftarrow \sum_{j=1}^{n_c} \max(0, (q(a)_j - c_j))$
    else
        $r \leftarrow \sum_{j=1}^{n_c} \min(0, (q(a)_j - c_j))$
    end
    /* Update the estimation of the student competence level for each KC
       */
    for $i = 1 \cdots n_c$ do
        $c_i \leftarrow c_i + \eta \cdot (q(a)_i - c_i)$
    end
    /* Update the estimation of reward for the pulled arm              */
    for $i = 1 \cdots n_p$ do
        $w_{i,a_i} \leftarrow w_{i,a_i} \cdot \exp(\gamma r / p_{i,a_i})$
        $w_i \leftarrow w_i / \left(\sum_{j=1}^{n_i^a} w_i\right)$
    end
end
```

4 Results

We used numerical values from the example of [3] for the difficulty of the exercises, the definition of the predefined sequence and the ZPD. In this example, the goal is to teach children how to decompose numbers while manipulating money. The KC are competences such as knowing what a decimal is, ability to sum integers, ability to deal with money notes or with abstract numbers. The exercises are parametrized with parameters such as whether the objective is spoken, written or both, whether there are decimals, whether the values dealt with are from a real money or abstract values... Our implementation can be found on [2].

4.1 One student

4.1.1 Experiment setup

We first present results on only one student. The experiment is the following: we consider a student with initial KC=0.7 in all competences but underrated by the ITS that believes $KC = 0.5$ at $t = 0$ in all competences. The number of rounds is set to 50 (at each round, only one exercise can be proposed). For the sake of clarity, we only focus on the progress related to the competence IntSum (ability to sum integers). We ran 100 simulations of this experiment to average results.

4.1.2 Interpretation of results

Figure 2 compares the true progress in competence IntSum between the predefined sequence, our method and random method.

The predefined sequence starts by proposing too easy exercises and that is why there is no progress until the 25th exercise. On the contrary, our algorithm (Exp3) proposes relevant exercises faster than predefined sequence by correcting its underestimation of the student level. However, we see that the progress is less brutal than for the predefined sequence because it is forced to keep exploring. Our algorithm clearly outperforms the random methods which needs twice as much iterations to reach the maximum competence level.

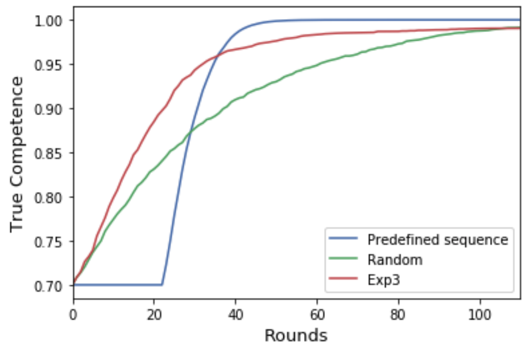


Figure 2: Comparison of the three methods on an experienced student

Figure 3 shows the evolution of both the true KC of the student and its estimate by the model. We initialized its value to a wrong one but one can see that the estimate quickly catch up with the true value and is a precise estimate in the end.

Figure 4 shows the regret for the random method and our method. Both methods are similar in the beginning when the algorithm has not learnt the recent reward from each arms and not corrected its estimation of competence. Then, the slope of regret diminishes for our algorithm because the estimation of KC is more accurate. It means that it starts proposing relevant exercises.

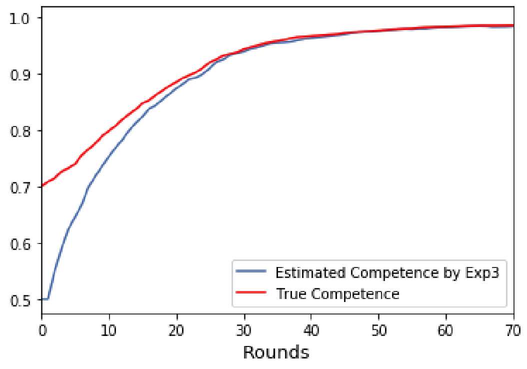


Figure 3: Evolution of the true KC and its estimated value

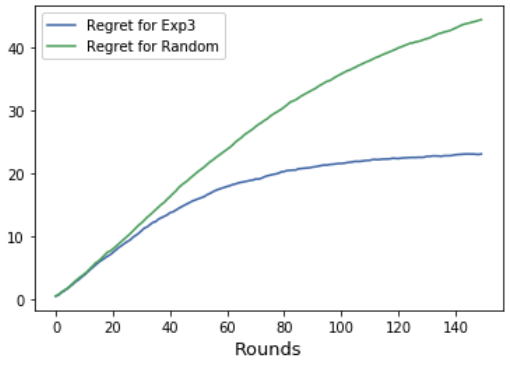


Figure 4: Regret curves for random and RiaRit methods

4.2 Q student

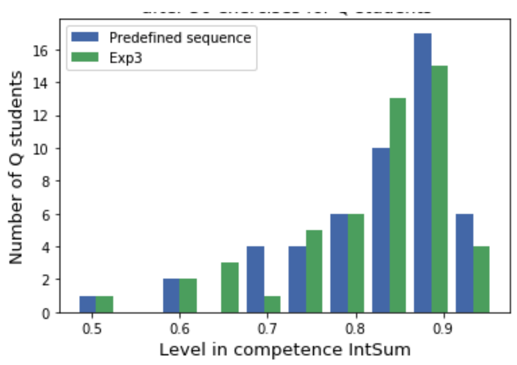
We then created a virtual class of 50 Q students. Their initial competence were initialized randomly following a gaussian distribution and their learning rate was drawn from a uniform distribution. Once again, the number of rounds is set to 50 and the results are averaged over 100 simulations.

Figures 5a and 5b show that our algorithm reach similar performances to the predefined sequence in term of learning progress. It is a good news since the prededefined sequence was designed specifically for this problem by an expert of education. The main advantage of our algorithm over the predefined sequence is its adaptability to different students level and learning rates. One way to see that is too look at the average number of failures during the simulation (Figure 5c). Our algorithm generates less failures because it adapts to the student level : the errors mainly comes from the beginning of the simulation, period when the algorithm learn how to esimate the level of the student. On the contrary, predefined-sequence is less flexible. In the sequence, the level of difficulty of each exercises is growing at the same pace for each competence but some students learn much slowly a specific competence so that this 'uniform pace' does not suit them. This explains why the number of failures is higher, the risk being that it could potentially generate frustration among students.

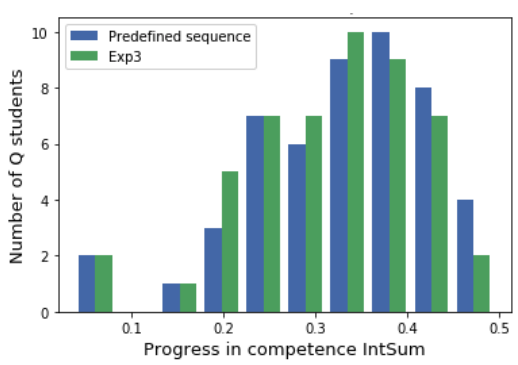
4.3 P student

We ran the same experiment but with P students instead. Each P student blocks on 5 exercises chosen randomly. It means that whatever their competence level, they will never succeed in these exercises.

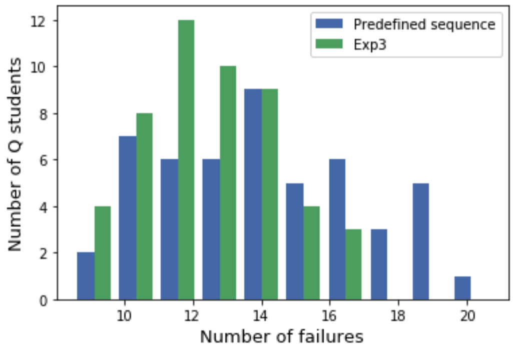
Figures 6a and 6b show that our algorithm provides better learning progress than the predefined sequence. Figure 6c proves that P students fail less with exercises proposed by the ITS than for those of the predefined sequence.



(a) Average maximum competence achieved



(b) Average progress made



(c) Average number of failures

Figure 5: Comparison of our method and a predefined sequence for 50 exercises on a class of 50 Q students

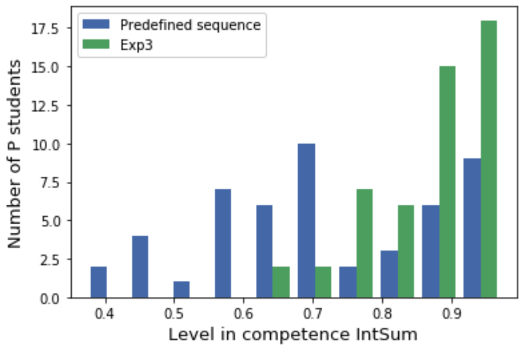
The first reason is that as seen before, it is able to adapt to each student difficulties. The second reason is that it can propose a larger variety of exercises. Indeed, while the predefined sequence contains only 10 different exercises (it's typically the case for a teacher), our ITS can propose as many exercises as the number of parameters combination (72 in our numerical simulation). Therefore, it is easier for our algorithm to overcome P students blocking difficulties.

5 Conclusion and future extension

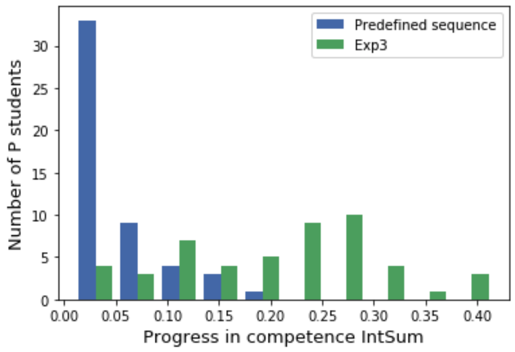
Our algorithm is able to personalize teaching sequences by estimating online students' competences and proposing a large variety of exercises in order to adapt to their level, learning rate and difficulties.

We have managed to reproduce the same results as [3] although our algorithm is slightly different from theirs (exp3 instead of exp4). Compared to a teaching sequence designed by an expert, our algorithm leads to similar performances on a population of students that have no limitation in the comprehension of activities (Q students) and significant improvement on students with a larger variety and stronger difficulties (P students).

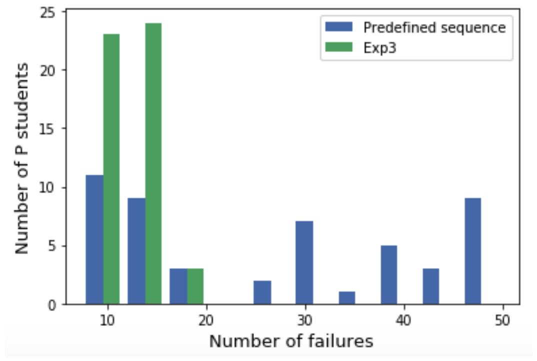
As future work, we can compare the results of their algorithm to ours on this experiment. In addition, it would be interesting to test other teaching scenarii to understand in which cases MAB algorithms can provide learning improvements.



(a) Average maximum competence achieved



(b) Average progress made



(c) Average number of failures

Figure 6: Comparison of our method and a predefined sequence for 50 exercises on a class of 50 P students

References

[1] Peter Auer et al. "The Nonstochastic Multiarmed Bandit Problem". In: *SIAM Journal on Computing* 32.1 (2002), pp. 48–77. DOI: `10.1137/S0097539701398375`. eprint: `https://doi.org/10.1137/S0097539701398375`. URL: `https://doi.org/10.1137/S0097539701398375`.

[2] Yousri Sellami François Darmon. *ITS*. `https://github.com/fdarmon/its`. 2018.

[3] Manuel Lopes et al. "Multi-Armed Bandits for Intelligent Tutoring Systems". In: *CoRR* abs/1310.3174 (2013). arXiv: `1310.3174`. URL: `http://arxiv.org/abs/1310.3174`.

[4] Anna N. Rafferty et al. "Faster Teaching by POMDP Planning". In: (2011). Ed. by Gautam Biswas et al., pp. 280–287.