INTELLIGENT VIRTUAL TUTOR FOR PROJECT-CENTRED LEARNING

Alessandro Scoppio, Georgios Patsiaouras, Khasbulat Kerimov, Camillo Rohe, Ibrahim Hadzic
Department of Science and Engineering
Maastricht University
a.scoppio,g.patsiaouras,b.kerimov,c.rohe,i.hadzic
@student.maastrichtuniversity.nl

June 27, 2019

ABSTRACT

Project-centred learning is a student-centred learning process that involves groups of students solving real-world problems. Nonetheless, when students lack of required prior knowledge, this process can result in worse overall performances. In this work, this problem is investigated in the domain of software programming and a solution is proposed: a system that assists students in acquiring necessary knowledge using Deep Learning and Intelligent Search to provide each of them with a set of programming exercises that maximize their learning path. The system adapts to each student by choosing the next exercise using the student current knowledge state, from which the best exercise is derived.

***K*eywords** Adaptive Learning System · Intelligent Tutoring System · Deep Knowledge Tracing · learning programming

1 Introduction

1.1 Context

The development of projects enables students to effectively gain valuable knowledge and experience. Throughout their development, students face a variety of challenges. Some challenges encourage

students to learn important concepts, others are simply a disadvantage. Especially in technical projects there are drawbacks due to a heterogeneous level of knowledge of the participating students. This knowledge gap makes group work unnecessarily complex in delegating tasks and resolving conflicts between group members.

The main objective of this project is focused on first year students of the Department of Data Science and Knowledge Engineering (DKE) of the University of Maastricht. Students are required to complete a technical group project. In order to complete the project successfully and on time, students must acquire certain skills in a short timeframe. The project requires an understanding of the concepts of software development and project management as well as a complex mathematical understanding.

1.2 Problem statement and Motivation

Experience shows that the requirements of the project in the first semester exceed the abilities of some students. This is likely to create a bad learning experience.

Generally, it is assumed that the performance of a student in the group project is related to the student's previous programming experience which differs substantially among students. Each student enters the Bachelor's program at DKE with a different background, school and educational level. As a result, the levels of knowledge of students are different, especially regarding programming skills.

A difference in programming skill causes the behavior known as 'Hero Programmer'. Here, a group member with better skills carries out the project independently. This poses a severe problem as students who are already familiar with programming become better, but at the same time students who have no or limited knowledge do not improve. Therefore, there is a danger that the knowledge gap among students will widen further during their studies.

In addition, the small time frame of the project does not encourage the creation of positive externalities in the sense that good programming students help the less capable ones. As said before the benefits of working in group projects include all these small details that help individuals grow inside of a team. As a result, the aforementioned problem attacks this principle directly by excluding specific members from a constructive learning experience.

In order to tackle this issue an educator needs at first to know the knowledge state of a student in programming. In principle, a high school diploma could provide insight into a student's cognitive and technical abilities. However, any such diploma usually does not cover programming skills.

Learning programming is an extensive undertaking as it requires the application of algorithmic thinking, understanding program execution, and remembering the syntax features of a programming language. Consequently, there is a need to evaluate students' programming skills and to ultimately improve these skills.

This inevitably leads to the next issue: estimating the students' ability to learn. The task of modelling and predicting human learning is influenced by different fields such as education, psychology, neuroscience and cognitive science. Learning is basically a reflection of human cognition, which involves a highly complex process. The ability to learn varies from student to student and must therefore be approached individually. In general, a better learning experience is achieved when the difficulty of a exercise matches the student's ability. On the contrary, if the exercise is too easy, the student is not challenged but bored but at the same time if the exercise is too demanding, the student becomes frustrated. Therefore, if the exercise involves a reasonable degree of difficulty, the student is more likely to be challenged and motivated [1].

A solution to evaluating and improving students' knowledge requires a modern and adaptive approach. As elaborated in the next section, this cannot be achieved with static learning methods. Therefore, the DKE needs to introduce a modern system that supports Bachelor students in mastering challenges in their group projects.

1.3 Research Questions

This project focuses on designing an adaptive system that provides students with the optimal set of exercises based on their skills.

The research questions are formulated as follows:

- Is there a method to assess the programming skills of students?
- Are students better prepared for future projects by providing them with a selected set of programming exercises in an automated way?
- What kind of algorithm will provide an optimal set of exercises?
- Can a dynamic system evaluate the skills of student while teaching them at the same time?

2 Related Work

Our research on this project is inevitably focused on the area of Intelligent Tutoring Systems (ITS). From this area, prominent systems have emerged that pursue different strategies to facilitate learning, such as avoiding syntax errors, displaying visual output, and providing tips. Theoretically, an ITS provides a dynamic and adaptive learning process that teaches the same principles to different students by adapting the learning process to the student. Conati defines an ITS as:

> *The interdisciplinary field that investigates how to devise educational systems that provide instruction tailored to the needs of individual learners* [2].

ITSs have been investigated in several studies with previous researches focusing mainly on computer-aided teaching systems (CAI). The simplest type of CAI system is generative. Generative CAI systems have been developed for situations that require repeated practice of a skill, such as vocabulary or arithmetic. Adaptive CAI, on the other hand, is a family of more complex algorithms that can take the information from the student to the selection of problems, or the course of tutorials. These systems can use simple techniques to estimate the skills of a student and select problems based on that estimate [3]. The principle of an ITS could be used to teach programming to new but also experiences users.

When it comes to teaching programming, there are many high-quality tutorials online that teach learners the introductory concepts of programming. These tutorials combine several strategies to support learning and motivation based on human needs, strengths, and weaknesses. However, they are not personalized but offer everyone the same sequence of exercises. For this reason, considering the situation of bachelor students in their first year, it is easy to conclude that such static techniques would not be effective.

By following an alternative concept, the task of providing the optimal set of exercises to students can be deduced to the task of estimating the probability of a student solving each task.

2.1 Deep Knowledge Tracing

Knowledge Tracing (KT) is a specific task in the domain of ITS. It can be formally described as the continuous assessment of a student knowledge **while** he/she is solving a sequence of exercises (hence a ***trajectory***). The history of the past submissions can give effective insights on how the student will perform in the next interactions. In KT, interactions take the form of a tuple of $x_t = (q_t; a_t)$ that combines the "question" or exercise being answered q_t with whether or not the

exercise was answered correctly a_t. A model then predicts whether a student is capable to solve an exercise.

In the past years, many approaches have been developed to solve the KT problem, such as using a hidden Markov model in Bayesian Knowledge Tracing (BKT)[4] or a standard logistic regression model in Performance Factors Analysis (PFA)[5].

Even though these two approach were promising, **Deep Knowledge Tracing (DKT)** outperformed them all, improving the predictions AUC of 10% and 20% on top of BKT and PFA respectively [6].

DKT predicts the probability of a student solving or not an exercise using a Recurrent Neural Network (RNN) such as Long Short-Term Memory (LSTM) [7], a deep neural network optimized to process sequential data. By using different sets of weights (or gates as defined in the original paper) it allow to model long time dependencies in the input data. This is applicable in various domain, ranging from time-series to signal processing, but the main area of use has always been Natural Language Processing (NLP). Being student submissions over time a sequential data, RNNs should be able to map the submissions history with the success of the next exercise, resulting as having as output at each time-step the probability of solving the next exercise.

In general, representation learning in the field of machine learning is the task of learning a model to create meaningful representations from low-level raw data inputs. The goal of representation learning is to reduce the amount of human input and expert knowledge needed to pre-process data before feeding it into machine learning algorithms [8]. This implies important advantages as explicit coding of human domain knowledge is not required. Consequently, DKT models are able to capture more complex representations of student knowledge. This includes the interdependence of different concepts and the ability to use information from an input in a prediction at a much later point in time. However, one disadvantage of RNNs over simple hidden Markov methods is that they require large amounts of training data.

Inside the DKT model, the input layer passes a vectorized question-and-answer interaction into the network to update the hidden state. The hidden states can be viewed as successive encoding of relevant information from past observations that will be useful for future predictions. The hidden layer refers to the latent encoding of knowledge state which is based on the current input and previous latent encoding of knowledge state. The output layer then entails the predicted knowledge state which consists of probabilities of a student solving the exercises. These probabilities are valuable because they serve as indicator of the student's learning progress. Whenever exercises were solved, new concepts were learned.

Multiple DKT implementation exists, and some of them perform better than others. In our work we replicated results from various of this implementation, coming up to the conclusion that the DKT+ [5] suited better our dataset. DKT+ introduces two regularization terms that assess two different problems in the original DKT formulation: *reconstruction* and *waviness*. The reconstruction problem happens when the prediction of solving the exercises that the student just correctly solved decrease instead of increasing, or vice-versa. The waviness is problem, on the other hand, appears when the knowledge state of the student is not consistent across time.

3 Methodology

As defined already in the previous section, the goal of this research is to design an algorithm capable of guiding students through the process of improving their programming skills in an adaptive and customized way.

3.1 Proposed architecture

The system consists in three modules. Each module acts as a micro-service that serves another one. Together, the modules form a system that interacts with students by evaluating and teaching programming skills by means of a tailored set of exercises to better prepare them for programming projects. The system can be described as follows:

- The **Tutoring module** provides an interface between the ITS and the student. Through this interface the user is able to attend the learning and evaluation process.
- The **DKT module** is responsible for handling the student submissions and evaluate the student knowledge state.
- The **Search module** provides the next exercise to be solved by the student using the knowledge state evaluated by the DKT module.

The overview of the architecture can be seen in figure 1. Each one of the modules and their interaction will be described in the following sections.

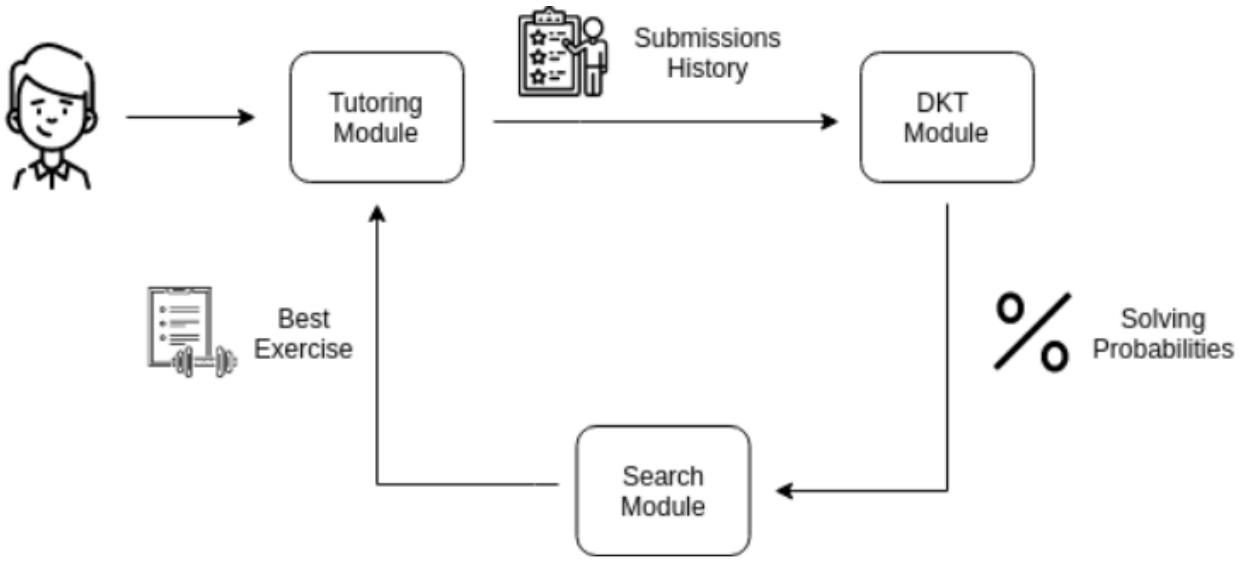

Figure 1: Architecture of the system and workflow.

3.2 Dataset

To investigate the feasibility of our method, an appropriate dataset needed to be found, as the ones used in the related work do not cover the domain of software programming. The dataset used, was provided for a coding contest and available to public by Hackerrank [9] and consists of roughly 286,000 anonymous submissions by 10,000 users attempting to solve the exercises. In total 2,865 exercises are presented in the initial dataset, comprising together 16 different categories such as exercises for learning C++, Java, Python, as well as Data structures, Databases, AI, Mathematics etc.

Content-wise, the exercises are supposed to prepare the students for their upcoming project. The project demands knowledge and skills in areas of basic programming, Java programming, and content related to the project. For instance, in the previous project 'graph coloring' knowledge about graph theory was required to succeed in the project. The exercises cover the required topics and vary in their difficulty-level and are labelled accordingly.

To compare the dataset size with the relative work in the DKT research, we tested the models also on a benchmark dataset, introduced in [4]. ASSISTments 2009u dataset was compiled from the ASSISTments Skill Builder Problem Sets, in which a student gains knowledge by working on similar exercises until he can correctly answer n in a row (where n is usually 3). This record contains 328,291 submissions (question-answer interactions); there are 4,417 student IDs and 124

skills. This means that in the dataset used in the referred research is less sparse, and every exercise has more submissions.

3.2.1 Data pre-pocessing

Having a dataset that consists of 2,865 exercises leads to an unnecessary complexity of a model, since the number of input nodes of a DKT and branching factor of a search algorithm are dependent on the number of exercises. In addition, some of the exercises were solved only by a few learners and some of the learners were solving only a few exercises. In order to reduce the complexity and to sharpen the modelling of individual exercises and students, the numbers of exercises and learners were decreased by leaving out the exercises and learners that were submitted less then a certain number of times. This pre-processing of the dataset involves a trade-off. While the number of exercises was intended to decrease, the number of submissions need to remain proportionally large. This trade-off let us evaluate different numbers of submissions per exercise and student to be removed. Eventually, the threshold of 20 submissions was selected. After pre-processing the number of exercises decreased from 2,865 to 1,377, while the total number of submissions decreased by roughly 50,000.

3.3 DKT Module

As previously stated, DTK allows to predict the probability of a student solving an exercise based on the previous submissions.

To train a DKT model, a set of submission $(q_t; a_t)$ is needed, where q_t is the ID of the exercise, and a_t is the indication whether the exercise was solved or not. These submissions need to be transformed into a fixed-length vector x_t since RNNs only accept fixed length of vectors as the input. Therefore, to convert student performance into fixed length of vectors a question is represented as a vector by using one-hot encoding. For one vector all elements are 0s except for a single 1 in the position of the question ID. When an exercise was solved by a student the corresponding answer label is also represented as an one-hot encoded vector of the question and when it was not solved simply as a zeros vector. Thus, the length of each input vector is two times the number of exercises in the data-set, while the number of time-steps is equal to the number of submissions. Once processed by the LSTM cell, the input is encoded in a latent state, that represent the latent knowledge state of the student. This latent state of knowledge is then passed to the output layer to calculate the probabilities of correctly answering each question. Recursively, the hidden knowledge state at time-step t will be used together with the next input to calculate the predictions in time-step

$t+1$. So, when there is a sequence of question-answer interactions of length T_i for the student i, the DKT model assigns the inputs $(x_{i_1}, x_{i_2}, ..., x_{i_{T_i}})$ to the outputs $(y_{i_1}, y_{i_2}, ..., y_{i_{T_i}})$ accordingly. The input vector undergoes a series of transformations inside the hidden layer, which captures useful information that is hard to human-engineer, and forms a sequence of hidden states $(h_1, h_2, ..., h_T)$.

Since the temporal relation between submissions is crucial in this domain and in general in sequential data, a custom batch generator was developed to be able to use the powerful Stochastic Gradient Descent algorithm, that approximate the gradient for mini-batch of samples at each iteration. Specifically, in our implementation the well-known Adam optimizer was used [10].

Extension of the model To further improve the results of our model an attempt was made of including additional information other than question and answer. The other information available in our were category and difficulty. Adding the category of the exercise in the interaction revealed to be not feasible, since the dataset contains different categories and sub-categories with exercises present in multiple ones. This would have "confused" the model instead of helping with a better prediction, therefore was not implemented. The difficulty, on the other hand, should be specific for each exercise and give very important insights in how the knowledge state of the student changed after this interaction. Following the work in [11] we concatenated the difficulty information at the tail of each interaction. Specifically, the information was originally a single value between 0 and 1, it was binned in 10 categories and then one-hot encoded as the question and answer. Surprisingly though, this resulted in a decreased accuracy as shown in the Experiments section.

3.3.1 Sequence constructor

The sequence constructor is the module of our application that completes the system as it is the main component responsible for identifying the best exercise to introduce to the student. Out of a set of exercises and a history of submitted exercises by a specific student (solved or not solved) an algorithm is need to determine which exercise, to be given next, would maximize the probabilities of solving other exercises and therefore optimising the learning trajectory of the student. As simple as this task may sound, it contains a hidden complexity focused around the fact that a student solving an exercise does not necessarily implies that the student gained any constructive knowledge from it. After careful consideration, we decided that, in order to preserve an interesting, feasible and educative learning process, the exercise to be chosen next should be derived by searching on all sequences of exercises.

The search algorithm selected for our problem is ExpectiMax. It is an algorithm capable of introducing modeled uncertainty during the process of searching. The uncertainty in this case can

be found in the response of the user, meaning whether the student solved or didn't solve the exercise that was given to him/her. The algorithm is by nature a search tree with consecutive *max* and *chance* nodes. The example of such search tree is presented on the figure 2.

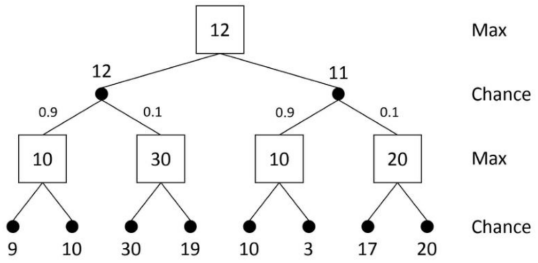

Figure 2: Example of the expectimax search tree. The value on the leaf nodes are propagated basing on the value and the nature of the parent node.

In every iteration of the algorithm and thereby going deeper in the search tree, all the possible exercises are considered to be chosen as the next one. There are two possible scenario after this: either the student solves the exercise or she doesn't. The chance nodes takes into consideration both of the outcomes by calculating the expected sum of the evaluations. More formally the value of a root node is calculated recursively as follows:

$V(s) = \max_a \sum_j p_{ij} V(s_{ij})$,where V is evaluation of the state and p is the probability of solving or not solving the exercise. Since the algorithm is by itself a tree search with depth d and branch factor of b, the time complexity of an algorithm is $\Theta(b^m)$.

Optimization

In order to optimize the time of search and to make the learning process more enjoyable we can specify the search window for an exercise, so the exercises is neither too easy nor too difficult [12]. Therefore we prune the exercises in the search tree with probability values less then 0.4 and more than 0.75.

Evaluation of the state

As the search algorithm is trying to maximize the values of the state of the student (e.g. learning ability or skill vector), we need to incorporate the evaluation of the state. And since the state of

the student is represented by the probabilities of solving, the evaluation can be represented by the following methods:

Average probability: Simple mean value of the probability of solving

Minimum probability: Minimal probability of solving the exercise is taken into the consideration, focused on improving the "weak" points of a student.

3.3.2 Tutoring environment

Following the original idea of creating a complete intelligent virtual tutoring system, we define the tutoring environment as the interaction interface between the system and the student. Via this interface the student, can solve the exercises that our system is providing and improve his/her skills in programming. For presentation and understanding purposes we see in a mock-up what the system would look like if it was implemented.

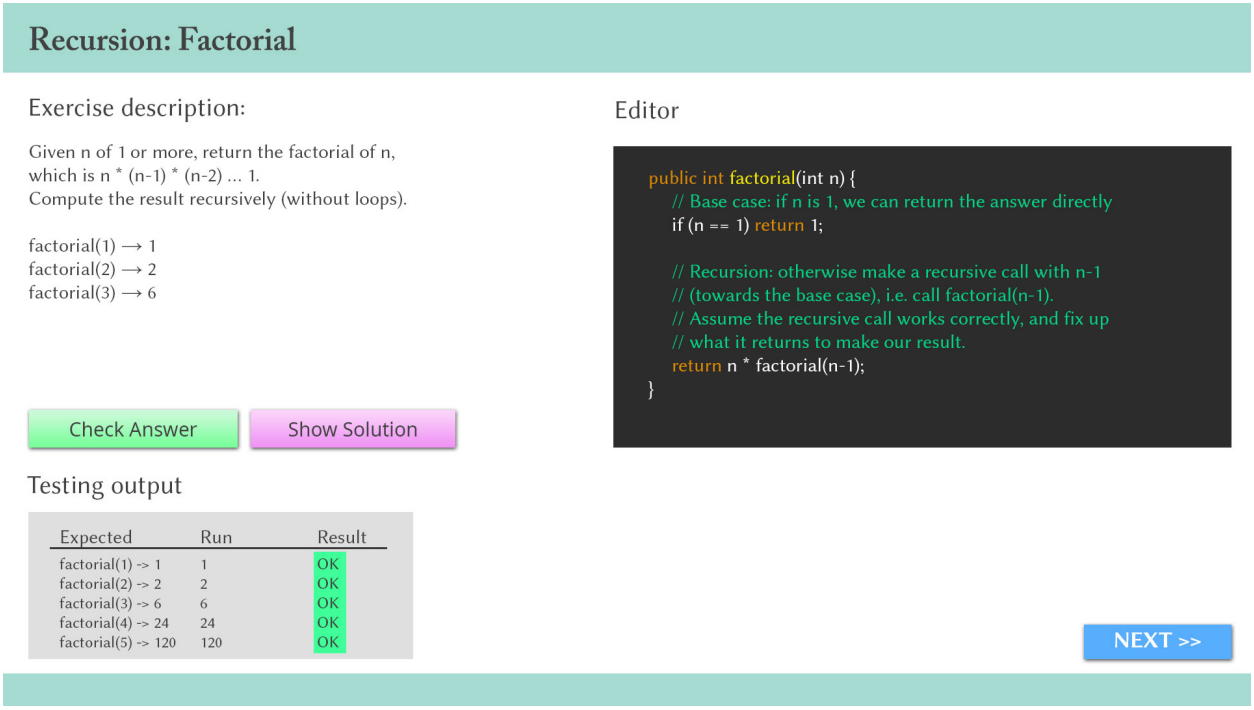

Figure 3: Mock up of tutoring environment

On the top left the description of the exercise is displayed that the student has to solve. At the top right is an interactive Java code editor that the student can use to enter the answer to the exercise.

Additionally, in the middle left of the platform, we see a button "Check Answer" which validates the answer. As described in the previous sections, the answer is evaluated by performing specific component tests and validating the output. In the example of the figure, the exercise is to build a recursive function for factorial calculation. The component tests include calling the function with different inputs. For each of the test cases, the input is validated and the student receives feedback about which test has failed so that he/she knows how to improve his/her code. If the student realizes that he/she cannot solve the exercise, he/she can switch to the next one by clicking on the button at the bottom right.

The tutoring environment is outside the scope of this project, thus it is presented in this section for the purpose of giving to the reader a more holistic and complete view of the project.

4 Experiments

In order to validate the performance of the approach proposed, different sets of experiments have been carried on. The two main modules of the system proposed - DKT module and Search module - have been tested individually before being merged. To validate the DKT module, the different implementations explored have been tested with both the benchmark dataset "ASSISTment 2009u" and on the investigated dataset "HackerRank".

	DKT +		**DKT**	
	Assist2009	HackerRank	Assistment2009	HackerRank
AUC	0.8243	0.8517	0.8238	0.8372
AUC(Curr)	0.9595	0.9650	0.8543	0.9117

Table 1: Performance of the DKT and DKT+ models on different datasets

The student performance predictions made by each model are presented in tabular form and accuracy has been assessed in terms of Area Under Curve (AUC). An AUC of 0.50 always represents the random score. A higher AUC score means higher accuracy. AUC refers to the prediction of the next exercise, while AUC(Curr) refers to the of the same time step. As visible, the regularization terms introduced by the DKT+ severly improve the AUC of the current preditction.

4.1 DKT Prediction visualization

After succesfully training LSTM on the dataset provided by hackerrank, we decided to investigate the dynamics of the evaluation of the students. In order to achieve this, we calculated the predictions

made by DKT for each student after each attempt and investigated closely the dynamics. Then we sorted the results by the score at after the last attempt.

The values on figures 6 and 8 is calculated as the average probability of solving the exercise. On the x-axis is the history of submissions and the color indicates whether the attempt on the x-axis was successful or failing.

Top evaluated students

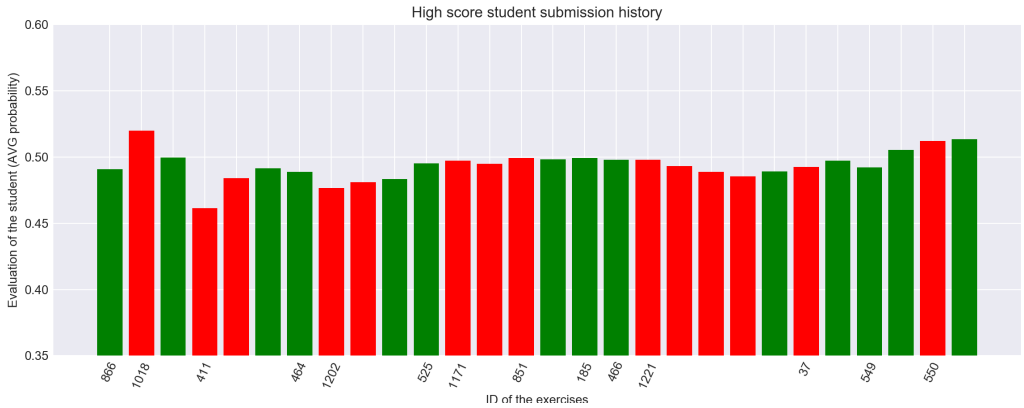

Figure 4: Top 1 evaluated student

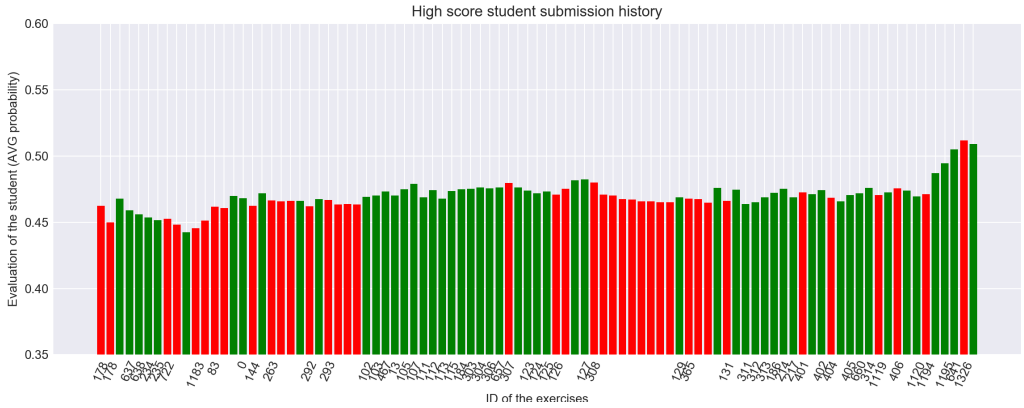

Figure 5: Top 3 evaluated student

Low evaluated students

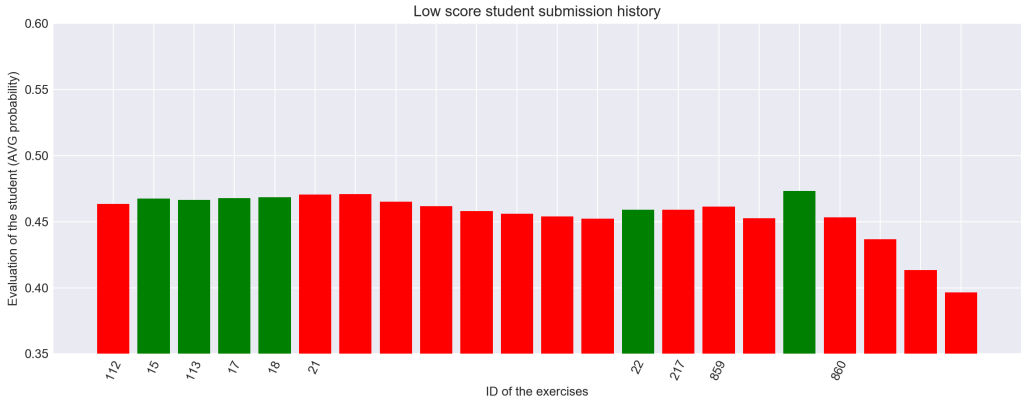

Figure 6: Low 1 evaluated student

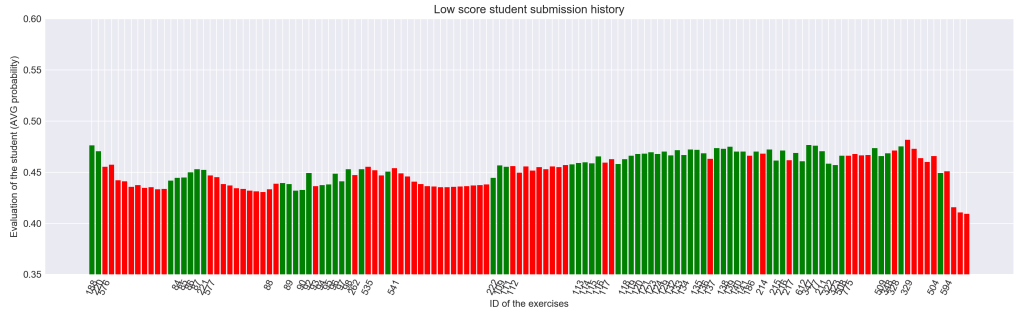

Figure 7: Low 3 evaluated student

As we can see from the figures in the experiment, the DKT module predictions are not entirely stable. If we look closer at the history of submissions of low performing students, we can see that the average probability of is decreased drastically on their last attempts. This is somehow counter-intuitive, since we expect that even failing attempts are supposed to help the student to learn. The issue of the unstable results was addressed in the research about DKT+ [5] and perhaps should be investigated more in the application of a deep knowledge tracing.

4.2 Expectimax investigation

The expectimax tree search is supposed to suggest the exercise that is going to increase the average probability of solving the most. We are going to investigate the exercises that are suggested to the

Top 3 student in his last submissions. In order to achieve this, we cut off last 5 attempts (right before the score has increased steeply) from his history and calculate the probabilites.

Results

For depth = 1, the suggested next exercise for a sequence is # 1054, while the actual exercise that was solved by student is # 1194. Although, these exercises are present in different categories, the exercises 1054 showed the greatest increase in the probabilities of solving.

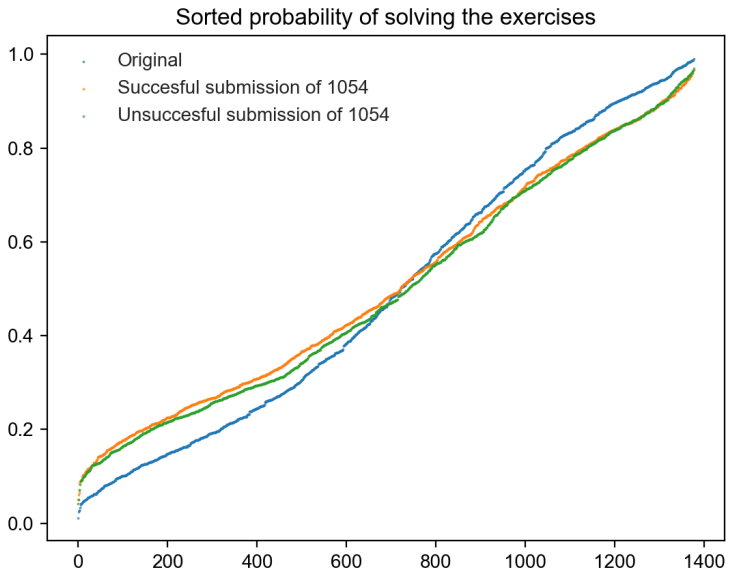

Figure 8: Probabilities of solving exercises arranged in the ascending order. On the X-axis is the order of the exercise. Yellow line represents the results after successful submission of the exercise and green line represents the results after incorrect attempt.

As we can see on the figure, the knowledge representation of the student has become more even distributed and the average probability of solving has increased comparing to original submission history. Unfortunately, search algorithm is highly computational expensive and for depth of 2 the running time of the algorithm with 1377 exercises was around 1 hour.

4.3 System validation experiment

In this section we propose the setup of the experiment that is suggested to be carried out with participation of the first year Bachelor students in order to finally answer one of our research

questions: "Are students better prepared for future projects by providing them with a selected set of programming exercises?"

We propose to split students into two randomly selected test groups. Both of the groups, before starting the first semester project will be asked via mail to participate in study in the system that runs on a web application. The students should be notified about the experiment and possibly rewarded for participation, although the results must not be evaluated, since it may affect the objectivity of the experiment. The groups can be introduced as:

- **First group** of the students is provided with the set of exercises, that are suggested by the ITS after each attempt. The first N exercises are handpicked and should not be very difficult to solve, e.g. have difficulty below 0.5, according to Hackerrank dataset.
- **Second group** of the students is supposed to solve the set of exercises, that are randomly selected from a set of exercises. These exercises should also not be difficult.

Since the purpose of the ITS is to prepare students for the first steps of the project, it is important to evaluate both of the groups after short period of time, e.g. after completing first period. The evaluation of the results of the experiments can be measured by comparing the grades for the programming courses in the first period of a study.

Alternative approach to system validation

The other possible method to validate the overall performance is to simulate training of the student by training image classifier. The problem would match the unique characteristics of our approach by training our algorithm to identify classes inside images. One image classifier would represent a user trying to solve exercises, but exercises on this domain would be images, and solving would be classifying. Our algorithm would join in, by providing the image classifier with the appropriate images to make it learn quicker and better in identifying. After elaborating more on this approach, we came to the conclusion that it might not be applicable to our problem and we decided not to continue with this idea inside the project.

5 Results and Discussion

As the result of our work we designed an algorithm that is theoretically capable of recommending exercises in an automated way, basing on the knowledge. We presented the trained DKT+ model as the application of this algorithm on the dataset of submissions provided by HackerRank alongside

with the implemented tree search algorithm. Each of the module was tested on the examples from the dataset and the results were presented. Together with tutoring module this system comprises the ITS that is supposed to recommend exercises to a student basing on his past performance.

The DKT model showed competitive performance on the programming exercises dataset, despite the increased complexity and sparsity of data. Nonetheless, we could indicate the limitation of the model, which results in decreasing probabilities of solving the exercise, after the exercise was already solved. Since the overall system is operating with the student knowledge representations, it is heavily dependent on the prediction of the DKT module, and thus a higher performance can be only achieved when the model is in align with reality. One of the possible solution is to further decrease the sparsity of data and try to achieve greater number of submissions per exercise.

Other limitation of the system is the running time of the algorithm. Since the search has complexity of $\Theta(b^m)$, with the number of exercises of 1377 the time required for one search for greater depth is too long, which makes it unfeasible to search for every attempt.

6 Conclusion

The path from pinpointing the problem to developing a system that recommends exercises was very interesting and challenging. We had to deal with several areas of research with a focus on intelligent tutoring systems and deep knowledge tracing. The discovery of new concepts in these areas inspired us to fully understand the modelling of the underlying problem. Several solutions were tested for effectiveness and applicability what deepened our knowledge of several machine learning concepts. After successfully modelling the knowledge state and progress of students, this project again highlighted the possibilities of applying neural networks to real-world cases.

6.1 Future work

As it was already stated, one of the main problems to solve in the ITS we proposed should be focused on tackling on limitations of the LSTM mechanism applied to the tasks of Knowledge Tracing. To to this, new and more advanced architecture for knowledge tracing should be considered. Furthermore, the algorithm could be enriched with the addition of student features like prior knowledge, completion rates, time on learning etc. More information about the student should result in having a better model of the progress and state. In addition, content features like problem difficulty, skill hierarchies etc. could be used for creating a context-dependent DKT implementation. That could take place by adding an extra input vector containing all of these features.

Although student and content features are an important addition a possible additional research could include information about the psychological state of the student. Using data of this kind, the algorithm could be able to choose best exercises based not only on the content but also on the mental state of the student during the learning process. This information could also be of help during the group formation or other aspects of the project.

Finally, during this research the validation of the system unfortunately, remained as an unwritten chapter as no valuable solution was found inside our time frame. A system evaluator would be of great value, because it will allow a better fine tuning of the system. Thus we believe that future work could include an implementation of this.

6.2 Social impact

For the following reasons it can be assumed that a successful implementation of the project can have a strong social impact. The ultimate aim of this project is to improve the effectiveness of the practical application of technical knowledge in group projects. On the way, a sufficient level of programming abilities among students needs to be created, which inevitably leads to a better learning experience in the first group projects. Because a project is more successful if the necessary technical requirements of the participants are fulfilled. It can therefore have a positive impact on a student's career, not only by encouraging students to complete their studies earlier, but also by increasing their qualifications through positive externalities and good work collaboration. This positive impact could be reflected by statistically significant higher project scores compared to a control group.

On the other hand, the educational process is the field characterized by personal interaction, while the application of the ITS itself assumes that it automatically offers tutoring. This could cast doubt on the effectiveness of such a method. It is equally likely that the students do not entirely agree with the proposed type of tutoring. Nonetheless, learning programming can generally be fruitful in the sense that it teaches a very pragmatic and logical way of thinking that one can use for other tasks. Finding suitable exercises can hereby be very motivating.

Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains. (B. Gates)

The proposed intelligent tutoring system backed with machine learning can easily scale to be applied to numerous students and even to other domains other than programming. Therefore it

could provide a high quality personalized learning experience, also outside of the project-centred learning domain.

References

[1] T. Effenberger, "Adaptive system for learning programming," p. 9, 05 2018.

[2] C. Conati pp. 2–7, University of British Columbia, Vancouver, 07 2009.

[3] R. Kass, "Student modeling in intelligent tutoring systems: Implications for user modeling," *User Models in Dialog Systems*, 01 1989.

[4] X. Xiong, S. Zhao, E. V. Inwegen, and J. E. Beck, "Going deeper with deep knowledge tracing," in *EDM*, 2016.

[5] C.-K. Yeung and D.-Y. Yeung, "Addressing two problems in deep knowledge tracing via prediction-consistent regularization," in *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, L@S '18, (New York, NY, USA), pp. 5:1–5:10, ACM, 2018.

[6] L. Wang, A. Sy, L. Liu, and C. Piech, "Deep knowledge tracing on programming exercises," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, L@S '17, (New York, NY, USA), pp. 201–204, ACM, 2017.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[8] A. C. Yoshua Bengio and P. Vincent, "Representation learning: A review and new perspectives," p. 1798–1828, 03 2013.

[9] HackerRank, "Hackerrank: Challenge recommendation." `https://www.hackerrank.com/contests/machine-learning-codesprint/challenges/hackerrank-challenge-recommendation`. Accessed: 25-06-2019.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[11] L. Zhang, X. Xiong, S. Zhao, A. Botelho, and N. T. Heffernan, "Incorporating rich features into deep knowledge tracing," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, L@S '17, (New York, NY, USA), pp. 169–172, ACM, 2017.

[12]