A Framework for the Estimation of Students' Programming Abilities

Ella Albrecht
Institute of Computer Science
University of Goettingen
Göttingen, Germany
ella.albrecht@cs.uni-goettingen.de

ABSTRACT

In times of increasing numbers of students and high usage of e-learning systems, student models are a good way to get an overview of what is currently occurring in the classroom, analyze students' behavior and estimate their learning progress. In our work, we develop a framework which estimates a student's programming knowledge by looking at his responses to open-ended programming assignments. The model we construct incorporates multiple applications of multiple skills in one exercise, multiple submissions and varying knowledge components involved in the same exercise.

1. INTRODUCTION

During the last years, the number of students has increased rapidly. Especially in introductory courses, hundreds of students are attending. This makes it infeasible for educators to take care of each student individually. On the other hand, to deal with large amounts of students, many institutes use e-learning and e-assessment systems to support their teaching. These systems allow large data collection on which data mining and learning analytics techniques can be applied to build student models. Student models are used to estimate a student's cognitive state, e.g., his/her motivation, knowledge, misconceptions or learning style and preferences [4]. A student model can be used to provide students personalized course material fitting to their current knowledge and learning habits. Furthermore student models can be used to predict student's performance and identify students which are at risk to intervene in a timely manner. Besides, we can use a student model to identify problematic course contents. This knowledge can be used as a basis for restructuring and redesigning the course.

In our research, we want to develop a framework for the estimation of student's knowledge regarding programming. Therefore, we look at students' solutions to open-ended programming exercises. For each exercise, it is defined which knowledge components (KC) are required to solve the exercise correctly. KCs describe the individual components of knowledge which are required to solve a particular task or problem. The task in an introductory programming course is to learn to write simple programs which meet the specifications given in text form, i.e., the exercise description. Therefore KCs can be, e.g., the programming language's constructs, i.e., syntax and semantics, correct usage of a compiler or IDE, error understanding and debugging ability, or the translation of specifications to program code. Then, it is checked whether the student has applied the KCs in his/her solution correctly. From theses observations a student model can be constructed which is able to estimate a student's knowledge state.

2. PROBLEM STATEMENT

Knowledge cannot be assessed directly, because there may be several reasons why a student made a mistake. For example, a missing `break` in a `switch-case`-block may be just due to sloppiness, because the student does not know the `break`-statement, or because the student does not understand how the commands in a `switch-case`-block are executed. Because of these uncertainties often probabilistic models are used for student modeling.

Bayesian Knowledge Tracing (BKT) [5] is one of the most widely spread student modeling approaches. It uses Hidden Markov Models to model students' learning. It was at first applied to programming exercises for LISP in the ACT Programming Tutor. The domain knowledge was represented by production rules of the form "to achieve goal X do Y" where Y may be a subgoal. The knowledge of a student was described as the probability that the student knows a rule. Since there was a deterministic order of which rules need to be applied to solve an exercise correctly, the student's knowledge could be estimated by looking at the student's solutions rules order. But in imperative or object-oriented languages like C, C++, or Java one can only extremely rarely define a deterministic order of statements.

Kasurinen and Nikula [7] have applied BKT on students' results to Python exercises. As domain knowledge they have defined guidelines for preferred solutions, e.g., each open file should be closed. Moreover, they have checked whether the student has used the guideline in his/her solution. However, the set of KCs was very limited.

Berges and Hubwieser [2] as well as Yudelson et al. [10] used the Rasch model from Item Response Theory (IRT) to estimate student's knowledge of object-oriented concepts in Java instead. In IRT, the relationship between responses to items, i.e., exercises, and a latent trait, i.e., an ability or KC, is described as a logistic function. Different from BKT,

it also takes the difficulty of an item into account.
BKT as well as IRT have the main drawback that they are single skill models, i.e., for each KC a separate model is constructed, and it is assumed that each exercise only requires one KC. For programming assignments, this assumption is of course not sustainable. Performance Factor Analysis (PFA) [8] is able to deal with multiple skills per exercise but as BKT and IRT also does not consider dependencies between KCs. However, in the programming domain there are dependencies between KCs, e.g., one needs to know how assignments or incrementing works when using a `for`-loop, or that the knowledge of a `while`-loop can influence the knowledge of a `for`-loop. It was also shown that integrating dependencies of knowledge into a student model can improve the model [3, 6]. Another special property of programming assignments is that KCs can be required multiple times in one exercise, e.g., if multiple loops are needed to solve the exercise. We also want to investigate the influence of substeps during the solution process to a model's accuracy. To the best of our knowledge, there does not exist a modeling approach so far which fulfills all of the requirements for programming assignments we have stated above.

3. RESEARCH METHODOLOGY AND APPROACH

Before we can make use of a student model in a course, several steps have to be taken. First, we need to identify what we expect the students to learn in our course, i.e., which KCs shall be acquired. In the first iteration of our research, the KCs we want to use for our model are the concepts of the programming language, e.g., `if`, `for`, variables, arrays etc., rules for good programming practice, e.g., each declared variable shall be used, allocated memory has to be freed, etc., as well as the fulfillment of the specifications by checking whether the program produces the correct output.
In a second step, we need to know which KCs are required to solve a particular exercise as we want to build our student model from the data we gain from their solutions to programming assignments. For example, summing up the numbers from 1 to 100 requires among others the knowledge of loops or recursion. This example also shows us, that it is actually not that easy to define which concrete concepts are really mandatory to solve the exercise as we could write a correct solution without knowing loops if we know recursion and vice versa. In our work, we develop a knowledge requirements model (KRM) which models required KCs related to language concepts for a particular exercise. The general mapping of language constructs, e.g., elements of an abstract syntax tree (AST), to concrete KCs has to be done beforehand by a domain expert. The KRM for a particular exercise is learned automatically from different correct solutions to that exercise based on their ASTs and structural analysis. We divide correct solutions into blocks and determine the set of KCs used in the block. From these sets we construct a tree where each path describes an alternative solution. By comparing a student's solution to the KRM, one can get the KCs which were applied correctly, incorrectly or are missing in the student's solution.
Despite the comparison with the KRM, we also use compiler and static analysis tool messages to assess the incorrect application of a KC, e.g., static analysis tools can deliver hints on, e.g., misunderstanding of control flow. Dynamic tests like unit tests, help us to evaluate a student's general program writing ability, i.e. whether a student is able to write a program which meets the specifications, i.e., does what it is intended to do.

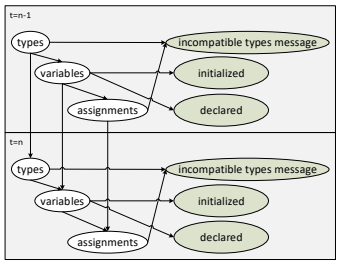

Figure 1: Example structure for a part of a DBN student model

The third step deals with the construction of the student model. We use Dynamic Bayesian Networks (DBN) for student modeling as they seem most appropriate to us. A DBN is a two-time-sliced Bayesian network where the state of a hidden variable depends on the states of the variables it depends on and the variable's state in the previous time step. Making observations in each time step updates the probability distribution of a hidden variable being in a particular state.
In our case, the hidden variables are the KCs, e.g., in Figure 1 the hidden variables (blank circles) are the concepts *types*, *variables*, and *assignments*. Observations in our student model are the results from the comparison of the student's solution with the KRM, compiler and static analysis tool messages as well as results from dynamic tests, e.g., in Figure 1 the observations (filled circles) are whether the student has declared and initialized a variable as well as whether an error message regarding incompatible types in an assignment appears. These variables can have the states *true* or *false*. With DBNs, we are able to deal with multiple KCs per exercise, their interdependencies, the uncertainty of which KC is affected by a certain observation and the uncertainty of which KCs are required to solve a particular exercise.
In our work, the structure of the DBN is defined manually by a domain expert. Though, one could also learn dependencies between KCs from data. The parameters of the DBN are learned from data using an expectation maximization algorithm with reasonable parameter constraints defined by an expert, e.g., limits for guess and slip probabilities. One problem that may occur, is that the parameter space is too large and we get computational problems when estimating the parameters of the model, if we use a very fine-grained KC definition. Therefore, we need to evaluate which granularity to choose to be able to estimate the parameters and still have an accurate model. Furthermore, we have to reason how to integrate multiple occurrences of the same KC in one exercise. Possible treatments are, e.g., majority vote or using uncertain evidences with a probability according to the ratio of correct/incorrect applications. We also want to analyze, whether multiple submissions, i.e., substeps preceding the final solution, improve the model.
In the second iteration of our research, we want to add further KCs which concentrate on more cognitive skills. The

first one is the debugging ability, which we want to assess by comparing two subsequent submissions when the first one indicates an error (or a failure) and check whether the problem was fixed.
As a further KC, we want to include variable roles [9]. Variable roles describe patterns of variable usage. They are defined by the successive values the variables obtain. An example for a role would be the most-wanted holder which is a variable that holds the best value encountered so far when going through a succession of values, e.g., when searching the smallest value in an array. The proper collocation of variable roles is essential for solving a task or achieving a goal in a program. Usually, students intuitively use variable roles in their programs. The lack of knowledge of a particular role could explain why a student may have problems to solve an exercise.
We want to evaluate our model by comparing it to common student modeling approaches like BKT, IRT and PFA.
In a last step, we want to analyze the model constructed from the data of our introductory C course to find out what students which are at risk have in common, which KCs seem most difficult to the students and how many exercises are required at least (on average, to reach a particular percentage of students) to gain sufficient knowledge in a certain KC.

4. CURRENT STATUS & NEXT STEPS

We have implemented a framework for the collection of metrics regarding students' solutions [1] which was successfully introduced in our introductory C programming course. It is mainly an e-assessment system where students can upload their solution and get some basic feedback. It collects compiler messages, results from static analysis tools, and results from dynamic tests to capture the correctness of the solution. In the first year, we got about 10,000 submissions of on average 250 students. We expect similar numbers this year.
Furthermore, we have identified the different KCs that we have in our course by going through the course material and previous programming errors of students. Based on that, we defined a hierarchical structure of KCs where the sinks are basic observations in form of rules like, e.g., the function returns a value if the return type is not void. We have also mapped compiler/static analysis tool messages to different concepts and implemented an AST parser. In a next step, we want to use the AST to filter the KCs from source code and construct our KRM.
Next, we plan to conduct a small case study with only a few KCs to evaluate the feasibility of our DBN student model.

5. EXPECTED CONTRIBUTIONS

In our work, we develop a framework for the estimation of students' knowledge regarding programming. One of our main contributions is the definition of a student model which has the following properties which are needed to construct the model based on solutions to programming assignments: multiple KCs per exercise are possible and their interdependencies are considered, uncertainty of affected KCs can be handled, individual KC requirements and usages can be treated, multiple submissions can be integrated, and a KC can be used multiple times in the same exercise.
Another contribution will be a KRM which is automatically generated from model solutions for each exercise and can be used to evaluate which KCs were applied correctly or incorrectly by the student.
Furthermore, we plan to not just look at language related KCs, but also more cognitive skills like, e.g., debugging ability. We hope that our model helps to get better insights into the learning process of students.
From the doctoral consortium we expect to get some feedback on our student model, especially hints for the evaluation w.r.t. metrics and data sets. We are also looking forward for further ideas for additional or alternative KCs which we can integrate in our model.

6. REFERENCES

[1] E. Albrecht and J. Grabowski. Towards a framework for mining students' programming assignments. In *2016 IEEE Global Engineering Education Conference (EDUCON)*, pages 1096–1100, 2016.

[2] M. Berges and P. Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 51–56, New York, NY, USA, 2015. ACM.

[3] A. Botelho, H. Wan, and N. Heffernan. The prediction of student first response using prerequisite skills. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, pages 39–45, New York, NY, USA, 2015. ACM.

[4] K. Chrysafiadi and M. Virvou. Review: Student modeling approaches: A literature review for the last decade. *Expert Syst. Appl.*, 40(11):4715–4729.

[5] A. T. Corbett and A. Bhatnagar. *Student Modeling in the ACT Programming Tutor: Adjusting a Procedural Learning Model With Declarative Knowledge*, pages 243–254. Springer, Vienna, 1997.

[6] Y. Huang, J. Guerra, and P. Brusilovsky. A data-driven framework of modeling skill combinations for deeper knowledge tracing. In *Proceedings of the 9th International Conference on Educational Data Mining EDM*, pages 593–594, 2016.

[7] J. Kasurinen and U. Nikula. Estimating programming knowledge with bayesian knowledge tracing. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 313–317, New York, NY, USA, 2009. ACM.

[8] P. I. Pavlik, H. Cen, and K. R. Koedinger. Performance factors analysis –a new alternative to knowledge tracing. In *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*, pages 531–538, Amsterdam, The Netherlands, 2009. IOS Press.

[9] J. Sajaniemi. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*. IEEE Computer Society, 2002.

[10] M. Yudelson, R. Hosseini, A. Vihavainen, and P. Brusilovsky. Investigating automated student modeling in a java MOOC. In *Proceedings of the 7th International Conference on Educational Data Mining EDM*, pages 261–264, 2014.