# Estimating Programming Exercise Difficulty using Performance Factors Analysis

Maria Tirronen Ville Tirronen

University of Jyväskylä
PL 35 (Agora)
Jyväskylä
Finland
40014
February 3, 2021

**Abstract**

This Work in Progress Paper studies student and exercise modelling based on pass/fail log data gathered from an introductory programming course. Contemporary education capitalizes on the communications technology and remote study. This can create distance between the teacher and students and the resulting lack of awareness of the difficulties students encounter can lead to low student satisfaction, dropout and poor grades. In many cases, various technological solutions are used to collect individual exercise submissions, but there are little resources for indexing or modelling the exercises in depth. Exercise specific feedback from students may not be easily obtainable either. In the present study, we attempt to create student-exercises models solely on pass/fail log data by using statistical techniques. We conclude that such data is insufficient for student modelling, but that it can be used to credibly estimate the difficulty of programming exercises.

## 1 Introduction

Many programming courses include online learning environments where students can complete exercises and write code. Such systems can practically ensure the functional correctness of student programs [1] and some of them

provide quality feedback on exercise specific items [2]. However, such systems, by design, reduce the dialogue between the teacher and the students by doing tasks that would have been previously carried out by the teacher (e.g., grading exercises). Relating this to the concepts of Transactional Distance Theory [3], automated systems are liable to reduce teacher-student dialogue and at the same time to increase the structure, or the inflexibility of the course, as the programs that assess student code do not often negotiate for partial credit. These factors increase the psychological distance between students and the teacher. For the students, this can amount to increasing learning difficulties, while, for the teacher, it can impede adjusting the course progression, correcting faulty exercises and evaluating student progress. Thus, there is a need for additional tools to help in monitoring the course and abridging the transactional distance by tailoring content to student progress.

In, general, it has been argued that the difficulty of programming exercises can be hard to assess and that the difficulty can be unintentional consequence of, e.g. the choice of required program constructs [4]. There are several approaches in the computing education community for providing tools to estimate exercise difficulties in automated fashion. For example, Ihantola et al. [5] attempt to derive exercise difficulty from observing student behaviour down to key press level detail. Similarly, Francisco and Paula [6] apply data mining techniques to estimate exercise difficulty using program metrics drawn from student submissions. Further, Alvarez and Scott [7] have studied the student perception of exercise difficulty and identified factors that contribution to it.

However, we believe that the exercise difficulty analysis has been most effectively tackled in field of intelligent tutoring systems (ITS). ITS employ precise modelling of exercises and construct statistical models of student performance (see e.g. [8, 9]). When course exercises are modelled with high precision, the precise modelling elements can be used to build statistical models of student performance and learning. Techniques such as Bayesian knowledge tracing [10], Matrix factorization techniques [11] and Learning Factors Analysis (see [12, 13]) can be used to model student knowledge and the difficulty of the learning items without any explicit feedback from the student. Further, such models can, and have been used to estimate difficulty of exercises and the difficulty program constructs required to solve them [14].

Although there are several high quality ITS systems available [15], they appear to be mostly employed in the institution from which they originate. Adopting new pedagogical tools and techniques is conditioned on teachers time and the confidence of successful implementation [16]. Further, computer science teachers are prone to perceive issues in making tools from different

institutions 'fit' their pedagogies [17], making elements such as ITS systems unlikely to be adopted if they require large upfront investment such as in depth modelling of exercises, or large changes in pedagogy.

In contrast to tutoring systems, which have a detailed model of the exercise structure (e.g., [8, 9]), many contemporary learning environments are built around test suites and student programs are constructed in a similar way that software is built in the industry. This is both convenient and cost effective, as is evident in a number of such systems in production today. Retrofitting student modelling on this kind of a system can be challenging. Instead of large number of clear knowledge components acquired from the precise modelling of the exercises, the log files for test suite based systems contain information on much less precise level. Often, such logs contain only 'pass/fail' information for submission attempts.

In this article we study application of statistical student models to coarse pass-fail data that is likely to be available from any learning management system. Although we found it unworkable to build a predictive student model based on such data, we were successful in building a credible model of exercise difficulties with it; the model and the teacher observations are well aligned. Due to simplicity and minimal requirements on the input data, we propose that, with some further work, this system would be simple, and unintrusive enough to be adopted by other programming educators.

# 2 Previous work

Modern student modelling approaches originate from the Carnegie Mellon Cognitive Tutors [8,10]. The Cognitive Tutors are a group of intelligent tutor systems that target multiple different fields such as basic level mathematics and programming. The Cognitive Tutors are modelled according to the ACT-R theory of human cognitive architecture [18]. In practice, Cognitive Tutors are formalized as production rule systems which guide the student in different tasks.

Student modelling became prominent alongside Cognitive Tutors, when Anderson et al. provided a Bayesian model of students performance. Since Cognitive Tutors were formulated as production systems, it was possible to precisely model student performance with individual and atomic, production rules [10]. After refinements, this model was able to predict student performance accurately [19]. Besides being used as a key part of ITS, the student model was also used to perform other studies, such as learning curve analysis [20].

The student modelling approach proposed by Anderson et al. is called

*Bayesian Knowledge Tracing* (BKT). BKT tracks student learning by modelling student knowledge in a Hidden Markov Model as a latent variable, student responses forming the output. The properties of BKT are well understood [21] and the method is commonly used in ITS.

Other student modelling approaches have appeared since. For our purposes one of the most interesting developments occurred in 2006 when Cen et al. proposed a method called Learning Factors Analysis (LFA) [22]. LFA consists of a logistic regression model augmented with a model searching technique for determining proper knowledge components for the model. The authors suggest that the heuristic search allows more precise tutoring as well as automatic discovery of domain models. The same article also introduces a simplified version of LFA, called the Additive Factors Model (AFM).

In 2009, Pavlik et al. proposed a method called Performance Factors Analysis (PFA), [13]. PFA is based on logistic regression similar to LFA and can be, considered as a further development of LFA. PFA adopts only the statistical model from LFA while the domain modelling heuristics are considered external to the method. As such, PFA is flexible and easy to adapt to different predictors. In later studies by Gong et al. PFA is also found to be more accurate than BKT [23]. Further, Gong et al. observe that PFA produces more plausible parameter estimates than BKT. Finally, Beck and Xiong claim in their study about limits of student modelling precision, that PFA is quite near to the maximum attainable in precision [24].

However, contrary to claim by Beck and Xiong, there are further accuracy gains to be had with PFA. In 2014, Galyardt and Goldin [25, 26] improve upon PFA by noticing that more recent data of student performance has more predictive value than older data (similar observation is also made earlier by Gong et al. [23]). Based on their observation, Galyardt and Goldin propose a method called R-PFA, which represents the practice history as a proportion of recent correct responses.

# 3 Modelling

For the purpose of model selection, we begin by comparing the Additive Factors Model (AFM) [22], Performance Factors Analysis (PFA) [27] as well as its recency-weighted modification [23] and the Recent-Performance Factors Analysis (R-PFA) [25]. We also include modifications of these models, S-only, R-only and R-AFM, introduced in [25], in our study.

AFM, PFA and their modifications aim to predict whether or not a student will answer an item correctly based on the student's history of practice. While AFM considers the total number of attempts, in PFA the number of

successes and failures are regarded as separate explanatory variables. The decay-weighted PFA represents the practice history as a recency-weighted number of correct and incorrect responses while in R-PFA the key idea is to represent the history as a proportion of correct responses.

In this study, the AFM reads as

$$\text{logit}(p_{ijt}) = \alpha_i + \beta_j + \gamma_j T_{ijt}, \tag{1}$$

where

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) \tag{2}$$

and

$$p_{ijt} = \mathbb{P}\big(X_{ijt} = 1\big) \tag{3}$$

with $X_{ijt}$ being a binary variable denoting a correct/incorrect response of the student $i$ at trial $t$ of the exercise $j$, while the quantity (3) is the probability of obtaining a success. The coefficients $\alpha_i$ and $\beta_j$ represent the initial student knowledge and exercise difficulty, respectively. The $\gamma_j$ represents the learning rate of given item. In (1), the variable $T_{ijt}$ denotes the count of past attempts (the sum of successes and failures) up to trial $t$.

Similarly, PFA reads as

$$\text{logit}(p_{ijt}) = \alpha_i + \beta_j + \gamma_j S_{ijt} + \theta_j F_{ijt}, \tag{4}$$

where $S_{ijt}$ and $F_{ijt}$ are the counts of past successes and failures, respectively. Here, the coefficient $\theta_j$ is to be interpreted as rate of learning from failed attempts.

In the model formulation by Gong et al. [23], a decay factor is introduced in (4). This factor updates the counts by decreasing the importance of prior performances. This is obtained by replacing $S_{ijt}$ by

$$S_{ijt}(d) = \sum_{p=1}^{t-1} d^{t-p} X_{ijp}, \tag{5}$$

where $d \in (0, 1]$ is a decay factor. Similar modification is also applied for $F_{ijt}$. The same value of $d$ is used for the counts of failures and successes.

Further, Galyardt and Goldin [25,26] use a proportion of recent successes in their R-PFA model, replacing (5) with

$$S^{\text{exp}}_{ijt}(d) = \frac{\sum_{p=1}^{t-1} d^{(t-p)} X_{ijp}}{\sum_{p=1}^{t-1} d^{(t-p)}}. \tag{6}$$

In (6), responses are weighted with an exponential kernel. While other kernels were considered by Galyardt and Goldin, the exponential kernel was found to give the most accurate results [26]. Further, we study a model in which the recent count of failures is also replaced by their proportion, $F_{ijt}^{\exp}(d)$. We follow the strategy used by Galyardt and Goldin [26] and compare R-PFA models with different decay factors for successes ($d_s$) and failures ($d_f$). To make a fair comparison between the models, we use different decay factors for successes and failures also in the PFA model.

Similarly to Galyardt and Goldin [25,26], we also consider a reduced PFA,

$$\text{logit}(p_{ijt}) = \alpha_i + \beta_j + \gamma_j S_{ijt}(d) \tag{7}$$

(S-only), and its counterpart in which $S_{ijt}(d)$ is replaced by $S_{ijt}^{\exp}(d)$ (R-only) as well as a modification of AFM where $S_{ijt}^{\exp}(d)$ is added to set of explanatory variables (R-AFM).

Finally, we introduce prior belief through ghost attempts (cf. [25, 26]). That is, we stipulate $g$ unobserved attempts $X_{ij(1-g)}, \ldots, X_{ij0}$ and set them to be incorrect. This corresponds to the assumption that at time 0, a student does not already master the exercise. Introducing ghost attempts does not have an effect on PFA but it enables computing $S_{ij1}^{\exp}$. We choose $g = 3$ that is smaller than the mean length of the non-empty practice histories in the data sets.

# 4 Adaptation to coarse pass-fail data

The aforementioned student modelling methods assume that each task represents a single, clearly defined Knowledge Component (KC), whereas common programming exercises are most often composed of a large number of intertwined concepts, or KCs [4]. Deducing KCs from either the functional characteristics of a student program or the exercise text is a non-trivial task. Multiple skills, such as mastery of programming language syntax or plan composition skills [28] are always in play. Regardless, there are attempts to automatically model the exercise content. These include mapping the student solutions into sets of ontological concepts [29] used as KCs and estimating KCs from syntactic elements present in the student submissions [14]. Nevertheless, implementing approaches like these require adopting programming language bound research software or implementing complex algorithms.

Since there are some hints that the student modelling systems presented in previous section perform better when provided a less complex model of exercise knowledge components [30], we propose to study simplified models in which each exercise forms its own KC. For observing exercise difficulty,

this choice can be justified with the assumption that, on average, all students have similar skills when they reach a given exercise. While this can obscure the difficulties faced by any individual student, it can effectively model the average difficulty of the exercises.

Additionally, our choice is motivated by high availability of coarse pass-fail data that can be used to build such models. If not ultimately accurate in modelling the student, the system is immediately adoptable on existing systems.

# 5 The data and course context

The data for this study was obtained from two eight-week elective introductory functional programming courses held during 2014 and 2015. Both course instances followed standard course model with lectures and supervised sessions, but also included a remote study option. The courses had programming exercises that were implemented in the web browser and the submitted answers were automatically assessed. These assessments form the data used in this article and we call these data sets as Func1 (2014) and Func2 (2015), respectively. Additionally, the courses followed a flexible format in which students could attain a partial credit by completing any number of the five available single credit modules.

The Func1 and Func2 data sets contain attempts by 129 and 179 students on 17 and 23 different exercises, respectively. The total number of trials are 4393 and 9070, and the total number of trials per exercise ranges from 77 to 469 and from 108 to 1019 in the Func1 and Func2 data sets, respectively.

# 6 Results

The models discussed in Section 3 were fitted using **glmer** function in the R package **lme4**. Both student and exercise effects in the models were fitted as random effects.

For the Func1 and Func2 data sets, we fitted the S-only, R-only and R-AFM models in the set $\{0.1, 0.2, \ldots, 1.0\}$ for the decay parameter $d_s$. The PFA and R-PFA models were fitted with both the decay parameters $d_s$ and $d_f$ in this set.

For the Func1 data set, the model with the lowest AIC [31] score (3692) was R-PFA with $d_s = 0.6$ and $d_f = 0.3$. For the Func2 data, the lowest AIC score (8183) was obtained with R-PFA with $d_s = 0.5$ and $d_f = 0.4$. However, for the Func1 data, four other R-PFA models and one PFA model reached

AIC scores that differed from the lowest AIC score by less than two, which indicates that there were no significant difference in performance between these models. Similarly, for the Func2 data, the differences in AIC scores between the model with the lowest AIC and five other R-PFA models and one PFA model were two at maximum.

The precision in predicting submission success or failure is low for all of the models. A low precision is to be expected due to requirement that the exercise is correct as a whole before marking it as a success. There are many ways for the exercise submission to fail and some of these are not related to the student skill level nor exercise difficulty: even professional programmers misspell names and forget parentheses [32]. Spurious errors add noise to the input data.

Although more practice, successful or unsuccessful, should increase the probability of a correct response [26], we found that most of the exercises had negative slopes ($\theta_j$) for the weighted proportion of failures. Also, for the Func2 data, the model with the lowest AIC score had one exercise with negative $\gamma_j$. The negative coefficients may be due to different input data than what is ordinarily used in student modelling systems. In our study, each exercise had a difficulty coefficient that was independent from other exercises and the exercises were composed of larger steps than what is usual with ITS. One could postulate that there were two modes for completing the exercises: easy initial success or starting off on a wrong path entirely. The effect would be similar to the one exhibited by our model; failures would not count as extra practice but as an indication that a wrong strategy has been chosen.

# 7 Practical results

We found that the exercise difficulty coefficients ($\beta_j$) correlated strongly with our experiences during the supervised sessions of the course. Manual inspection of student session logs was also in line with the model estimates. The exercises with low $\beta_j$ values did cause lot more issues to students than those with higher values. Low $\beta_j$ sessions were also rife with errors and awkward solutions.

Figure 1 shows the exercise difficulty coefficient ($\beta_j$) of each exercise in the order the exercises appeared during the course. The plot forms a 'difficulty curve' for the course, indicating difficulties experienced by the students at the various points of the course.

The shape of the 'difficulty curve' (Figure 1) reveals problems with several of the exercises in forms of unexpected peaks and valleys in the graph.

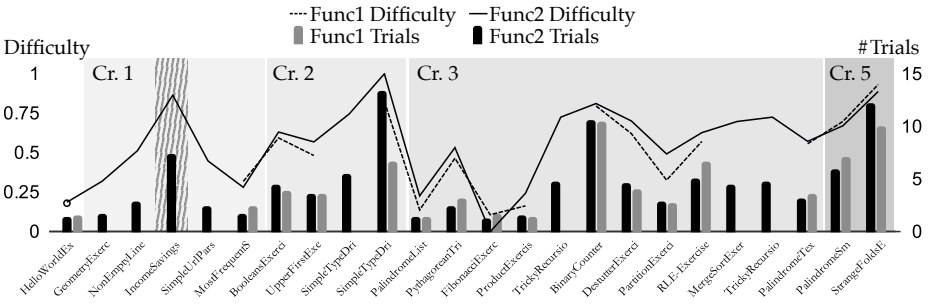


Figure 1: The estimated exercise difficulties ($\beta_j$) for each exercise ($j$) scaled to range [0,1] along with the average number of trials before success. The exercises appear in the same order as they were presented to the students during the course and the shaded boxes indicate the credit level of the exercises. The missing values on Func1 indicate exercises that were added between the two courses. The exercise that was removed between the courses is not displayed. The the stripes indicate a broken assessment program.

For example, the *IncomeSavings* (bolded in the figure) exercise had a misbehaving assessment system. We noticed minor difficulties with this exercise during the supervised sessions but the majority of the students worked independently and had greater difficulties without supervision.

The interpretation of the student model is complicated by the different submission rates between students. As observed by [33], where one student will submit the exercise repeatedly after every minor change, other may submit it only once or twice with large changes in between. Although the model partially accounts for the different rates, the student performance coefficients ($\alpha_i$) are thus not commensurate. Further, there is no data for the more difficult exercises by students who opted for partial credit, leading to inflated performance co-efficients for those students. As the result of these difficulties, we did not observe a correlation between the performance coefficients and the number of credits earned by students. However, the observed performance of students during supervised sessions did not seem a reliable indicator on how far the students progressed, either. We believe that external factors, such as workload from other courses, were a more deciding factor regarding the number of credits attained than observed proficiency in course exercises.

# 8 Discussion

We fitted PFA and R-PFA models and their modified versions to a pass/fail log file data obtained from two instances of an introductory functional pro-

gramming course. Although student models are usually fitted to a more precise data, we found that the resulting models provide credible information on the exercise difficulties, but not for student outcomes on the course. We suspect that the student models obtained are reasonable, but due to differing submission rates, they are not commensurate and thus cannot predict student outcomes.

Several avenues of improvement suggest themselves based on this experiment. Naturally, modelling the exercises with more precise KCs would likely bring advantages, but we conjecture that so would small changes to the exercises and student instructions. For example, to normalize submission rates, the students should be guided towards repeat submissions with small changes that aim to keep the code in state where it compiles, which is a good practise in general as well. Similarly, the exercises could record partial successes such as moving from non-compiling state to a compiling state, or moving from state with fewer passed tests into a state with more passed tests, which would provide more information for the model. Finally, following the argument by Luxton-Reilly et.al. [15], the modelling could be focused to exercises with the number of concepts (or KCs) limited to few at a time. Such exercises may also prove to have other pedagogical value.

In practice, we have found that the modelling of exercise difficulties has resulted in improvements to the course content. It has demonstrated malfunctioning assessment systems and weak spots in the course material. The obtained model was also used to re-arrange the exercises in to providing for smoother student experience in future instances of the course. At this point of the study, it could be argued that simply looking at average exercise scores or number of trials could have led to the similar conclusions. However, the modelling approach allows incorporating more information in the future and possibility of obtaining a predictive student model as well.

# References


[1] C. Benac Earle, L.-Å. Fredlund, and J. Hughes, "Automatic grading of programming exercises using property-based testing," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2016, pp. 47–52.

[2] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 15–26, 2013.

[3] D. Keegan, *Theoretical Principles of Distance Education*. London: Taylor & Francis e-Library, 2005.

[4] A. Luxton-Reilly and A. Petersen, "The Compound Nature of Novice Programming Assessments," in *Proceedings of the Nineteenth Australasian Computing Education Conference on - ACE '17*. Geelong, VIC, Australia: ACM Press, 2017, pp. 26–35.

[5] P. Ihantola, J. Sorva, and A. Vihavainen, "Automatically detectable indicators of programming assignment difficulty," in *Proceedings of the 15th Annual Conference on Information technology education*. ACM, 2014, pp. 33–38.

[6] R. E. Francisco and A. P. Ambrosio, "Mining an online judge system to support introductory computer programming teaching." in *EDM (Workshops)*, 2015.

[7] A. Alvarez and T. A. Scott, "Using student surveys in determining the difficulty of programming assignments," *Journal of Computing Sciences in Colleges*, vol. 26, no. 2, pp. 157–163, 2010.

[8] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive tutors: Lessons learned," *The journal of the learning sciences*, vol. 4, no. 2, pp. 167–207, 1995.

[9] A. Mitrovic and B. Martin, "Evaluating the effect of open student models on self-assessment," *International Journal of Artificial Intelligence in Education*, vol. 17, no. 2, pp. 121–144, 2007.

[10] J. R. Anderson, F. G. Conrad, and A. T. Corbett, "Skill acquisition and the lisp tutor," *Cognitive Science*, vol. 13, no. 4, pp. 467–505, 1989.

[11] N. Thai-Nghe, T. Horváth, and L. Schmidt-Thieme, "Factorization models for forecasting student performance," in *Educational Data Mining 2011*, 2010.

[12] H. Cen, K. Koedinger, and B. Junker, "Comparing two IRT models for conjunctive skills," in *International Conference on Intelligent Tutoring Systems*. Springer, 2008, pp. 796–798.

[13] P. I. Pavlik Jr, H. Cen, and K. R. Koedinger, "Performance factors analysis–a new alternative to knowledge tracing." *Online Submission*, 2009.

[14] K. Rivers, E. Harpstead, and K. R. Koedinger, "Learning curve analysis for programming: Which concepts do students struggle with?" in *ICER*, 2016, pp. 143–151.

[15] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "Introductory programming: A systematic literature review," in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, pp. 55–106.

[16] L. Ni, "What makes CS teachers change? Factors influencing CS teachers' adoption of curriculum innovations," *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 544–548, 2009.

[17] R. B.-B. Levy and M. Ben-Ari, "Perceived behavior control and its influence on the adoption of software tools," in *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, 2008, pp. 169–173.

[18] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind." *Psychological review*, vol. 111, no. 4, p. 1036, 2004.

[19] A. T. Corbett, J. R. Anderson, and A. T. O'Brien, "Student modeling in the ACT programming tutor," *Cognitively diagnostic assessment*, pp. 19–41, 1995.

[20] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User modeling and user-adapted interaction*, vol. 4, no. 4, pp. 253–278, 1994.

[21] B. van De Sande, "Properties of the bayesian knowledge tracing model," *JEDM-Journal of Educational Data Mining*, vol. 5, no. 2, pp. 1–10, 2013.

[22] H. Cen, K. Koedinger, and B. Junker, "Learning factors analysis–a general method for cognitive model evaluation and improvement," in *International Conference on Intelligent Tutoring Systems*. Springer, 2006, pp. 164–175.

[23] Y. Gong, J. E. Beck, and N. T. Heffernan, "How to construct more accurate student models: Comparing and optimizing knowledge tracing and performance factor analysis," *International Journal of Artificial Intelligence in Education*, vol. 21, no. 1-2, pp. 27–46, 2011.

[24] J. Beck and X. Xiong, "Limits to accuracy: how well can we do at student modeling?" in *Educational Data Mining 2013*, 2013.

[25] A. Galyardt and I. Goldin, "Recent-performance factors analysis," in *Educational Data Mining 2014*, 2014.

[26] ——, "Move your lamp post: Recent data reflects learner knowledge better than older data," *JEDM-Journal of Educational Data Mining*, vol. 7, no. 2, pp. 83–108, 2015.

[27] P. I. Pavlik Jr, M. Yudelson, and K. R. Koedinger, "A measurement model of microgenetic transfer for improving instructional outcomes," *International Journal of Artificial Intelligence in Education*, vol. 25, no. 3, pp. 346–379, 2015.

[28] E. Soloway, "Learning to program= learning to construct mechanisms and explanations," *Communications of the ACM*, vol. 29, no. 9, pp. 850–858, 1986.

[29] R. Hosseini and P. Brusilovsky, "Javaparser: A fine-grain concept indexing tool for java problems," in *CEUR Workshop Proceedings*, vol. 1009. University of Pittsburgh, 2013, pp. 60–63.

[30] Y. Huang, Y. Xu, and P. Brusilovsky, "Doing more with less: Student modeling and performance prediction with reduced content models," in *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 2014, pp. 338–349.

[31] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 199–213.

[32] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' build errors: a case study (at google)," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 724–734.

[33] M. Yudelson, R. Hosseini, A. Vihavainen, and P. Brusilovsky, "Investigating automated student modeling in a Java MOOC," *Educational Data Mining 2014*, pp. 261–264, 2014.