Stereotype Modeling for Problem-Solving Performance Predictions in MOOCs and Traditional Courses

Roya Hosseini, Peter Brusilovsky
University of Pittsburgh
Pittsburgh, USA
{roh38,peterb}@pitt.edu

Michael Yudelson
Carnegie Mellon University
Pittsburgh, USA
yudelson@andrew.cmu.edu

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@helsinki.fi

ABSTRACT

Stereotypes are frequently used in real life to classify students according to their performance in class. In literature, we can find many references to *weaker students*, *fast learners*, *struggling students*, etc. Given the lack of detailed data about students, these or other kinds of stereotypes could be potentially used for user modeling and personalization in the educational context. Recent research in MOOC context demonstrated that data-driven learner stereotypes could work well for detecting and preventing student dropouts. In this paper, we are exploring the application of stereotype-based modeling to a more challenging task – predicting student problem-solving and learning in two programming courses and two MOOCs. We explore traditional stereotypes based on readily available factors like gender or education level as well as some advanced data-driven approaches to group students based on their problem-solving behavior. Each of the approaches to form student stereotype cohorts is validated by comparing models of student learning: do students in different groups learn differently? In the search for the stereotypes that could be used for adaptation, the paper examines ten approaches. We compare the performance of these approaches and draw conclusions for future research.

KEYWORDS

Individual differences, Java, MOOC, Student modeling

1 INTRODUCTION

In the field of user modeling, it is common to distinguish stereotype user models and feature-based user models from one another [4]. Stereotypical user models attempt to cluster the multitude of users of an adaptive system into several groups (called stereotypes) that are considered to have similar needs in the sense of adaptation. The adaptation mechanisms treat all users who belong to the same stereotype in the same way. In contrast, feature-based models attempt to model specific features of individual users such as knowledge, interests, and goals. During the user's work with the system, these features may change, so the goal of feature-based models is to track and represent an up-to-date state for modeled features and use it for adaptation. Stereotypical user modeling is one of the oldest approaches to user modeling. It was originally developed by Elaine Rich [21] and was extensively used in many early user-adaptive systems [14]. However, over the years, feature-based user modeling approaches became dominant in almost all types of adaptive systems. With their better ability to represent individual users, feature-based models empowered many advanced personalization approaches. For example, in the area of adaptive educational systems, it has become common to represent domains to be learned as a set of knowledge components (KCs) and to independently model a learner's knowledge of each of these KCs. This leads to sophisticated knowledge modeling approaches, such as Bayesian knowledge tracing [6] that, in turn, has enabled high-quality prediction of student problem-solving performance and various personalization approaches.

Surprisingly, once online learning was scaled up to thousands of learners in modern massive open online courses (MOOCs), stereotype-based modeling was brought back to the forefront. We can cite many recent papers that mine MOOC log data in search of stereotypes that group users with the same behavior [1, 15, 23, 26, 27]. This work follows the same expectations as the early work on stereotypes in user modeling field: to make MOOCs adaptive, all users that belong to the same stereotype are expected to receive the same treatment from the system. So far, the work on stereotypes in these MOOC contexts has demonstrated some good results in predicting MOOC dropouts and failures. It does show that stereotypes could be useful for detection and possible prevention of these key MOOC problems. Could we deduce that further research on MOOCs will herald a major comeback for stereotype-based modeling? On one hand, the remarkable scale of MOOC data and new approaches to mining these data might open a way to more reliable stereotype construction that differs considerably from expert-defined stereotypes employed in the early days of user modeling. These stereotypes could potentially work much better by competing (or even winning) against feature-based models. On the other hand, current work on stereotypes and prediction in MOOCs has predominately focused on predicting coarse-grained (course-level) behavior, such as failure or dropout. It is not evident that stereotypes could be useful for predicting finer-grained problem-solving behavior, given that each course can feature many dozens of problems or exercises to solve.

In this paper, we have attempted to explore the prospects of stereotypes in MOOCs "beyond dropouts" – for predicting student performance at the problem level. We used data from a programming MOOC that included a large share of problem-solving activities and provided fine-grained data about user problem-solving behavior. Our goal was to find stereotypes that could be useful (or actionable) for predicting a user's success at solving problems. In

UMAP '17, July 09–12, 2017, Bratislava, Slovakia

DOI: http://dx.doi.org/10.1145/3079628.3079672

this context, "useful" means that problem-solving performance predictions would be different enough between stereotypes to enable personalized guidance to direct users to the most useful learning content. If useful stereotype-level models are found, then it is possible to use stereotypes for problem-level personalization; i.e., to predict problem-solving performance independently for each stereotype and use it to offer different interventions for different stereotypes. For example, all students within a given stereotype could be switched to a new topic once a chance to solve problems correctly for the current topic becomes high, or remedial material could be offered if the chance to solve a problem is too low.

The paper presents our attempts to find actionable stereotypes. Section 2 presents the context of our work. Section 3 explains our dataset, followed by Section 4, which elaborates on our assessment methodology. Section 5 reports our attempts to use simple demographic stereotypes, while Sections 6–7 present our search for more reliable behavior-based stereotypes. Sections 8–9 explain our findings on the behavior-based stereotypes. Surprisingly, despite our intermediate success in finding interesting behavior-based stereotypes, none of the stereotypes explored in this paper appeared to be truly "useful". Section 10 summarizes our results and discusses outcomes. We believe that our data points to a need to use finer-grained feature-based user models to support performance prediction and personalization for individual problems.

2 RELATED WORK

2.1 Student Behavior Analysis in MOOCs

Due to a large volume of available data and a surprisingly low completion rate, the analysis of student behavior in MOOCs emerged as an important topic just a few years ago. Perhaps one of the very first studies on MOOCs and behavior was the work in [3] that focused on the amount of time that students spent on various activities, as well as on demographic information about the students. In a more recent attempt [1], a taxonomy of individual learner behaviors was developed to examine the different behavior patterns of high- and low-achieving students. Another attempt was the work of [26], which adopted a content analysis approach to analyze students' cognitively relevant behaviors in a MOOC discussion forum and explored the relationship between the quantity and quality of that participation with their learning gains. In a similar attempt, [23] presented a hierarchy to categorize MOOC students into different engagement groups, based on their styles of engagement.

Overall, past studies have generally focused on resource usages, such as viewing course lectures, quizzes, assignments, and discussion forum activities to find the behavior of different groups of students and attempt to relate those behaviors with high and low levels of learning. However, there is some evidence from past work that demonstrates that focusing solely on resource usage might not lead to a reliable method to separate weak and strong students [5].

Unlike the past studies, the current work analyzes student behaviors by finding micro-patterns in student problem-solving activities, rather than by examining resource usage. Two similarly-minded attempts can be found in [24], which focused on the search for problem-solving strategies, and [27], which defined study habits by mining student navigation. However, neither of them explored behaviors by closely examining how students solved problems.

2.2 Assessment Data Analysis in Programming

Analyzing student solutions to programming assignments has received much attention during the past years. Recent work has used submission data to reveal multiple correct and incorrect ways to solve the same problem [9, 13], build an intelligent scaffolding system [22], model student knowledge in a program development context [20, 29], predict student grade [16], and understand student coding behavior through conceptual analysis [12].

The current paper contributes to the existing body of literature on analysis of assessment data by using compilation and submission data collected from students' problem-solving activities in a Java MOOC to understand (1) individual patterns of problem-solving (coding) behavior; (2) the impact of discovered behaviors on student performance in the programming course; and (3) any implications of the behaviors for accurately modeling student knowledge.

3 DATA

The data for the study comes from four introductory programming courses and MOOCs offered at a research-oriented University in Europe in 2014 and 2015. A single iteration of the programming course lasted for seven weeks and used Java as the programming language. Each week, students worked on tens of programming assignments with varying complexity. Less complex assignments were given when a new topic or construct (e.g., loops) was introduced, and as students created a number of smaller programs with those constructs, they moved to larger assignments that required the use of multiple constructs. The students worked on the assignments in the NetBeans environment. The assignments were downloaded into the programming environment through the Test My Code-plugin [25], which was used to assess the students' code automatically, as well as to collect data from the programming process of those students who consented to the use of their data for research purposes.

The collected data included key-presses with time, assignment information, and student id, and was aggregated to describe meaningful events in the students' programming process. The events used for this study were *running the program*, *running the tests for the program*, and *submitting the program for grading*; also, the first five generic events (inserting or removing text) were included for each assignment to make it possible to analyze transitions to meaningful events. For each event, information on program compilation and correctness were extracted for the data in a posthoc analysis using JUnit test sets, and finally, *programming concepts* for each problem-solving state were extracted using JavaParser [11].

Students were given a demographic questionnaire that solicited their age, gender, programming background, and the highest level of education attained. Out of 2739 students that started the courses, 1788 students were included in the initial sample (the cutoff was 2500 recorded events, which corresponds to roughly a 33^{rd} percentile of the first week of the course workload). Out of those, 798 students answered the questionnaire and were included in the final analysis sample. Table 1 shows key statistics for all participants.

4 THE ASSESSMENT APPROACH

In this work, we attempted to determine whether separating students into various cohorts for the group-based adaptation would be useful. In particular, we are interested whether we could find

	Student sample		Age	Gender	Programming background	Prior education
Course	*Initial*	*Final*	*min / mean / max*	*M / F / NA*	*None / Some / More*	*Pri.&Sec. / College / Grad*
MOOC 2014	1286	90	16 / 32.3 / 65	90% / 9% / 1%	4% / 81% / 15%	63% / 12% / 25%
Traditional 2014	263	192	18 / 23.2 / 44	62% / 38% / 0%	59% / 38% / 3%	78% / 3% / 19%
MOOC 2015	984	372	15 / 30.8 / 66	77% / 22% / 1%	30% / 60% / 10%	58% / 13% / 29%
Traditional 2015	206	144	19 / 24.3 / 45	63% / 35% / 2%	56% / 39% / 5%	74% / 3% / 23%

Table 1: Background information on the students that took the courses.

groups of students that are so distinct that their members learn differently. As a criterion to judge whether we were able to obtain the desired split between groups when looking at multiple ways to group students, we used differences between *models of student learning* in each group. Our rule of thumb is that if groups are truly different in how their members learn, the group models would demonstrate different performance in a cross-prediction task.

Before turning to the innovative approaches to separate students by their programming behavior, we demonstrate our evaluation approach by assessing simpler ways of grouping. These simpler groupings would include those known *a priori* (e.g., demographics, prior achievements) and those known *a posteriori* (e.g., overall course performance). By comparing innovative approaches to the simpler ones, we also monitor whether our behavior-based approach differs enough from existing approaches. Naturally, we would prefer behavior clustering results that do not align with simpler groupings. After evaluating existing simpler approaches to student grouping, we will examine student clustering using programming behavior mining. All approaches will be validated using groups/clusters models of learning and predicting across group/cluster boundaries.

4.1 Modeling Student Learning

To model student learning, we used an approach called performance factors analysis [18] that is based on logistic regression. PFA represents student abilities as a random factor θ_i, concept difficulties as fixed-factor intercepts β_k, and concept learning rates from correct and incorrect submissions as γ_k and ρ_k, respectively. Equation (1) shows the canonical form of the PFA. Here, σ is the inverse logistic function, while s_{ik} and f_{ik} are the counts of the prior student's successful and failed attempts to apply concepts.

$$P(Y_i = 1|\theta, \beta, \gamma, \rho) = \sigma\left(\theta_i + \sum_k (\beta_k + \gamma_k s_{ik} + \rho_k f_{ik})\right) \quad (1)$$

Our choice of model was based on the compensatory nature of the PFA – multiple concepts used in student's submissions, together, form a cumulative signal that results in the observed outcome. The other candidate modeling approach – Bayesian knowledge tracing [6] – is not intended for multi-concept student transaction data.

We have made two modifications to the PFA, both of which improved the overall fit. First, we have switched from concepts defined across problems to within-problem concepts. Second, we $log(x+1)$-transformed the concept opportunity count. Both of these modifications proved to be useful in the work by Yudelson et al. [29], which considered data from the same source. The modifications only changed the scope of concepts and the way opportunity counting is done, while the canonical form of PFA remained the same.

In contrast to [29], we pre-processed the data differently. First, every snapshot of the student code was treated as an atomic unit of data. It was deemed successful if all tests passed; otherwise, the problem attempt was unsuccessful. Second, we only considered snapshots where students were testing, running, or submitting their code – i.e., purposefully checking it for correctness. Intermediate snapshots were not considered for student modeling. Consecutive testing, running, or submitting the code without modifications in between was treated as one attempt. Third, we considered all concepts that were used in student's code. Only considering changes in concepts with or without special treatment for removals (as in [29]) led to model performance degradation under our data preprocessing setup. Finally, we considered only students for whom we had background information (798 out of 1788 students).

Due to our modifications, the upper boundary for the number of PFA concept parameters went from 143*3-1 = 428 to 143*240*3-1 = 102,959. However, because of the problem-concept matrix sparsity, the actual number of parameters was 13542*3-1 = 40,625. Also, given the size of the data (about 392,000 student submissions), it was not possible to use conventional statistical packages. We used a modified LIBLINEAR tool [7]. The modification[1] was in the form of an additional solver that allowed grouped L^2-penalties to approximate random factors. The modified LIBLINEAR retained the ability of the original version to tackle large datasets successfully.

4.2 Comparing Student Models across Groups

The primary goal of this work was to find at least two groups of students in our sample that have *different models of learning*. Following the approach piloted in [28], given the breakdown of a student sample into n groups/clusters, we sub-sampled each group 20 times to extract 80 students as a training set and 20 students as a test set. Sub-sample models were built from each of the 80-student training set. We then used each of the $n * 20$ models to predict n corresponding sub-samples: one sample matched the group that the model was built on, while the rest were from the other group(s). Finally, we plotted n^2 model accuracies (means and standard errors), n of which represented model performance within the group, while the rest were between the groups.

Our criterion for students in different groups *learning differently* was that within-group model performance would be visibly better than between-group model performance. In the case where $n > 2$, that should be true for at least two groups out of n. An example of the *ideal* separation is shown in red in Figure 1. Here, a model built on group A is superior to the model built on group B when predicting test data from group A. At the same time, when predicting test data of group B, model B wins on *its own ground*. Thus, we say that

[1] https://github.com/IEDMS/liblinear-mixed-models

models *A* and *B* are sufficiently different since they prevail on the student strata they were built on and forfeit on other student strata. An *expected* case is marked in blue. We previously discovered this phenomenon in [28]. In this case, there is a domination of one model over the others, irrespective of the origin of the sub-sample. Such cases are marked in blue (model *B* vs. model *C*). Finally, a *sub-optimal* case of model *A* vs. model *C* (marked in green) occurs when one model wins on *its own ground* (here, *A*) but does not have an edge over or loses to the other model (here, *C*).

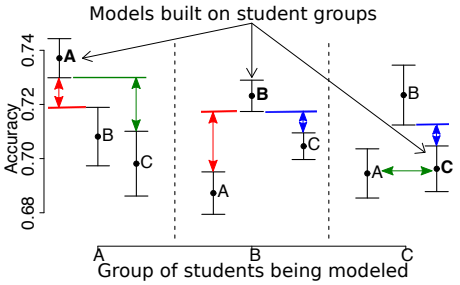


Figure 1: Between-group student model prediction accuracy differences (means and standard errors). Red arrows mark an *ideal* case, blue – *expected* case, green – *sub-optimal* case.

5 SIMPLE STUDENT GROUPING

As an example of our assessment approach, we first examined simpler demographic- and course performance-based approaches to student grouping. For the demographic data, we used gender and education level reported in the background information. Performance data were extracted from the course statistics available at the end. These groupings are summarized in the top five rows of Table 2.

Gender. Students were split by gender. The majority of students were males (about 71%).

Education level. Students were split into three groups. There were 524 students that had primary and secondary education, 154 students who attended college, and 120 students in graduate school.

Number of transactions. Students were split into three equal percentile groups – low, medium, and high – by the total number of problem attempts. When we employed a similar approach to investigate student groupings in [28], a subset of students that yielded more data produced a globally superior model as well. This grouping serves as our check for that phenomenon.

Problems Solved. This grouping was produced by an agglomerative clustering of four course-level counts: problems solved (at least one submission 100% correct), problems partially solved (at least one submission > 0% correct), problems attempted but not solved (at least one submission of 0% correct), and problems not attempted. The clustering yielded three groups: low (mostly not attempting problems), high (mostly solving problems), and medium (everyone else). This grouping is an overall student performance measure.

Percent Correct. This grouping was a split with three percentile groups with low, medium, and high values of overall percentage correct of the times students purposefully tested, ran, or submitted their code. This grouping separates students by their diligence.

Approach (no. features)	*Cluster sizes*	Prediction diff. *Score*	Prediction diff. *Cluster to note*
Gender† (1)	570, **228**	0.33	Female
Edu.‡ (1)	524, 154, 120	0.00	
#Trans.‡ (1)	266, 266, **266**	0.67	High
P.Solv.‡ (4)	218, 316, 264	0.00	
%Corr.‡ (1)	266, 266, 266	0.00	
C1 (45)	**383**, 415	0.67	1
C2 (245)	**416**, 382	0.67	1
C3 (245)	258, **158**, 382	0.67	2
C4 (245)	**295**, 503	0.67	1
C5 (245)	389, **272**, 137	0.67	2

† Male and Female; ‡ These groupings have 3 levels: Low, Medium, and High

Table 2: Approaches to clustering students. Top rows indicate simpler groupings while bottom rows (C1-C5) indicate advanced behavior-based clustering approaches.

Our preference for the cross-prediction group separation is in the order mentioned in Section 4.2: *ideal*, *expected*, and *suboptimal*. For simplicity's sake, we scored both group and cluster separations. A score of 1.0 would mean that the separation is *ideal*, a score of 0.67 would mean that the separation is *expected*, a score of 0.33 would mean that the separation is *suboptimal*, and otherwise score is 0.00. Cross-prediction differences between simpler groupings are addressed in the top five rows of the "Prediction diff." columns of Table 2. Out of the five simpler grouping approaches, only two had a non-zero score. In the case of gender contrasts, a model of female students had the edge over the model of male students when predicting the test data of female students. When predicting test data of male students, both models performed the same. Concerning the total number of student transactions grouping, the model of students contributing the most data had an edge. In fact, it was better than others, no matter what the test data predicted.

6 BEHAVIOR MINING

The key idea behind our behavior mining approach is to characterize student problem-solving behavior on the level of micro-patterns that define how the student progresses to the correct solution through several incorrect solutions, and how his or her knowledge grows from assignment to assignment. To build the micro-patterns, we started by processing student intermediate programming steps that classified the programming behavior at each step (section 6.1). Then, we applied sequential pattern mining to extract sequential micro-patterns (section 6.2). Next, the most frequent micro-patterns were used to build a profile vector (we call it a genome) that represented student problem-solving behavior. The stability of the behavior vector built from micro-patterns was checked to ensure the validity of our approach to mining problem-solving behaviors (section 6.3). Each of these parts is explained in more detail below.

6.1 Processing Intermediate Steps

To determine student problem-solving behavior, we started by looking into how students progressed in coding their problem solutions. We used *snapshots*, intermediate programming steps that were captured from student coding activities. Each snapshot recorded the

submitted code and its correctness on a suite of tests designed for each problem. As in [12], to mine programming behavior, we first examined conceptual differences between consecutive snapshots – i.e., we observed which concepts were added or removed on each step, and inspected how these changes were associated with improving or decreasing the correctness of the program. For simplicity, the conceptual difference was approximated as the numerical difference between the number of concepts in two snapshots. The procedure for mining the behaviors included two main steps: (a) labeling sequence of students' snapshots in each problem, and (b) mining micro-patterns of frequent behaviors (we call them genes), by conducting sequence mining on all of the labeled snapshots.

To label the sequence of student snapshots in a particular problem, the snapshots that were captured for the student in that problem (including generic, test, run, and submit snapshots) were collected and ordered by time. Each snapshot in the sequence was labeled based on the change in the *programming concepts* and *correctness* from the previous snapshot. The previous snapshot for the first snapshot in the sequence was defined as snapshot ∅, where the code has no concepts and passes no tests. Table 3 lists the labels that we used during labeling snapshots. The zero correctness value is to distinguish the snapshots where no tests were passed.

	Concepts		
Correctness	*Increase*	*Decrease*	*Same*
Increase	a	b	c
Same	d	e	f
Decrease	g	h	i
Zero	j	k	l

Table 3: Labels for encoding behavior in a snapshot.

As an illustration, assume that we have two snapshots for a student. The first snapshot has 10 concepts and passes half of the tests, while the second has 20 concepts and passes all of the tests. To label the first snapshot, we compare its number of concepts and correctness to the snapshot ∅ that has zero concepts and correctness. Since both the number of concepts and the ratio of passed tests increased in the first snapshot, it would be labeled as "a". The label for the second snapshot would be "a" too because the student added more concepts and increased the ratio of passed tests to one. As a result, the sequence of student snapshots would be labeled as "_aa_" – obtained by concatenating the labels of each individual snapshot. The"_" symbol marks the start and the end of the sequence.

Additionally, to distinguish between short and long steps (which is an important aspect of problem-solving behavior), another dimension could be added to each label to convey the extent of time that was spent on a snapshot. Since different students might have different speeds at programming, it is reasonable to use individualized thresholds for classifying the time spent on a step as short or long. This way, a snapshot would be labeled as short or long, depending on the time being shorter or greater than the median of time distribution for each student. In our coding, lowercase letters (a–l) represent *short* steps, and uppercase letters (A–L) represent *long* steps. For example, assuming that the median of time distribution for the student in the aforementioned example is 10 minutes, and that the student spent 15 minutes to develop the code in the first snapshot and another 2 minutes to make the minor change in the second snapshot, then the sequence of her snapshots would be labeled as "_Aa_".

We labeled 137,504 sequences of snapshots that were contributed by 1788 students solving 241 problems. The length of sequences ranged from 1 to 475, with an average of 5.3; 92,549 sequences had more than one step, 64,328 had more than two steps, 48,195 had more than three steps, and 38,768 had more than four steps.

6.2 Mining Problem-Solving Micro-Patterns

We mined frequent sequential patterns in students problem-solving sequences using an implementation of the SPAM algorithm [2] offered by the SPMF Library [8]. The input data to the SPAM consisted of 9254 sequences with at least two steps in them. SPAM discovers all frequent sequential patterns that occur in more than *minsup* of students' sequences. In this work, we chose a small *minsup* (e.g., 1% and 5%) to capture the patterns that are less frequent and may occur only in small groups of students. Also, no gap was allowed in SPAM to force the discovered patterns to have steps that appear consecutively in students' sequences. Finally, the length of the patterns was limited to two or more, as we were interested in observing how students progressed in their code in consecutive steps. SPAM discovered 245 frequent programming patterns occurring in more than 1% of students' sequences that were labeled with respect to change of concept, correctness, and time spent on a snapshot[2]. The top 20 common patterns and their frequency of occurrence are provided in Table 4.

Patterns 1–5	*Patterns 6–10*	*Patterns 11–15*	*Pattern 16–20*
AA (19.78%)	DD (8.53%)	Ac (6.24%)	Jc (6.04%)
AD (13.75%)	aA (8.31%)	DA (6.22%)	_Af (6.01%)
_AA (12.17%)	Ad (8.26%)	_AD (6.18%)	dD (5.61%)
Aa (9.75%)	Af (8.15%)	Dd (6.15%)	DE (5.57%)
AA_ (8.69%)	JJ (7.22%)	JA (6.13%)	Jj (5.45%)

Table 4: Top 20 frequent programming patterns occurring in more than 1% of students' problem-solving sequences. The numbers in parenthesis are the occurrence frequencies.

6.3 Using Micro-Patterns to Model Behavior

We used the micro-patterns discovered by sequential pattern mining to build individual behavior profiles as frequency vectors that showed how frequently each micro-pattern from a discovered set of 245 appeared in a given student's problem-solving behavior. The frequency vectors were normalized to add up to 1 in each vector. This approach was first introduced in [10], where it was used to find problem-solving behaviors in parameterized exercises. Following this paper, we also call the micro-pattern-based student profile the *problem-solving genome*.

To ensure that the vector of micro-patterns frequencies can capture stable characteristics of the student (i.e., it is as stable as a real

[2]Assuming that all sequences have an average length of 5, the maximum number of patterns that could be found has an order of magnitude of 8: There are 24^5 possible sequences that could be obtained from 24 labels (a–l, A–L), and the number of possible substrings in a 5-character sequence is $5 \times (5+1)/2 = 15$. Thus, the total number of patterns that could be found from sequences with a length of 5 is $24^5 \times 15 = 119{,}439{,}360$.

genome), [10] suggested to check the stability of the vector by splitting student sequences into two "halves" and to build the student's behavior vector from each of the two halves. If the two half-vectors (i.e., profiles built separately from each half of split data) of the same student were closer to each other than to half-vectors of other students, then we have strong evidence to claim that the behavior profile vector (genome) is stable. Following this suggestion, we split student sequences in two ways: once by randomizing sequences and dividing them into halves (random-split), and once by ordering sequences by time and dividing them into early and late halves (temporal-split). Then, we built behavior vectors for each half and calculated pairwise distances between the first and second half-vector of the same student (self-distance), and first/second half-vector of the student with first/second half-vectors of other students (others-distance). The distance between half-vectors was calculated using Jensen-Shannon (JS), as it is a common measure used for computing distance between frequency distributions.

We performed a paired Wilcoxon signed rank test to compare values of "self-distance" calculated from the first and second half-vectors of the same student to values of "others-distance" calculated from the first/second half-vector of the student with first/second half-vectors of other students. We found the random-split self-distance ($Mean = 0.349, SE = 0.003$) to be significantly lower than others-distance ($Mean = 0.659, SE = 0.001$), $p < 0.001$. Similar results were obtained with the temporal-split, while the self-distance in the temporally split half-vectors ($Mean = 0.425, SE = 0.002$) was larger than randomly split half-vectors, it was still significantly smaller than the others-distance ($Mean = 0.653, SE = 0.001$), $p < 0.001$. These observations supported the stability of using micro-pattern frequencies to represent student's problem-solving behavior. Also, the behavior profiles obtained with the proposed approach uniquely characterized student's problem-solving behavior and set them apart from the others.

Once we established stable vector-based profiles of student behavior, our next step was to use the micro-pattern representation of problem-solving behavior to group students based on their problem-solving styles. The next two sections explain the behavior groups that we found and their impact on student performance.

7 BEHAVIOR-BASED GROUPS

7.1 Clustering Students into Behavior Groups

To identify similar problem-solving behavior groups, we built behavior vectors of micro-patterns frequencies for each student and clustered students by using these vectors. To build the behavior vector, we used the problem-solving sequences of each student, obtained from all of the problems that they attempted to solve during the course. Each sequence represented consecutive snapshots that were captured while students were developing the program as a solution for a problem. We tried five different settings for clustering students behavior (see Table 5), changing the clustering method (hierarchical, spectral), and the number of clusters (2,3). We made sure that cluster labels in the advanced student groupings (C1-C5) did not have a significant overlap with the simpler groupings of students (discussed in Section 5) or between each other. C2-C5 labeled the snapshots based on concepts, correctness, and amount of time that a student spent on the snapshot, while C1 did not consider time. Also, the number of micro-patterns used in the labeling process differed: 45 patterns that were used in building behavior vectors in the first setting were obtained by setting SPAM *minsup* to 5%. The number of patterns in the rest of the settings was 245, which were obtained by setting SPAM *minsup* to 1%.

Approach	*#Patterns (Minsup)*	*Clustering Method (#Clusters)*	*Time*
C1	45 (5%)	Hierarchical (2)	
C2	245 (1%)	Hierarchical (2)	✓
C3	245 (1%)	Hierarchical (3)	✓
C4	245 (1%)	Spectral (2)	✓
C5	245 (1%)	Spectral (3)	✓

Table 5: Settings of the advanced clustering approaches.

7.2 Interpreting Discovered Clusters

In this section, we examine the nature of behavior-based grouping in greater detail. To make the differences clearer, we use settings with two clusters. As the clustering demonstrated, all three two-cluster settings separated students into two similar groups: one group with more constructive building steps, and one group who often massaged the code (i.e., added/reduced concepts without increasing the code correctness), and/or struggled in consecutive steps with no success. The settings with three clusters yielded a similar grouping for students as well, except that it separated a third group who had mixed behaviors as being closer to other two clusters in a subset of micro-patterns. As an example, Figure 2 illustrates the behavior groups that we obtained by spectral clustering with two clusters (Table 5, row 4). The Y-axis shows the ratio of occurrence of the top 20 micro-patterns for the two clusters. These patterns are re-ordered by the absolute difference between the two clusters.

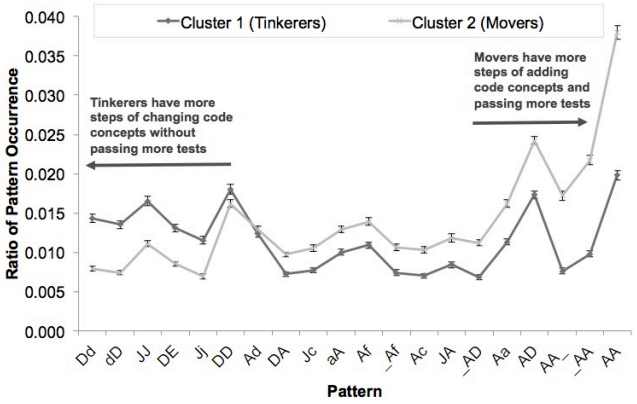


Figure 2: Top 20 programming patterns and their ratio of occurrence in each cluster from clustering approach C4. Patterns are ordered by the absolute difference of ratios between Cluster 1 (tinkerers) and Cluster 2 (movers). Error bars show standard error of the mean. The lines are added to distinguish the points that belong to different clusters.

As the figure shows, the groups differ by the frequency of micro-patterns on the extreme sides of the plot. As the left side shows, students in Cluster 1 have a higher frequency of "tinkering" patterns

(*Dd*, *dD*, *JJ*, *DE*, *Jj*), while the right side shows that the students in Cluster 2 demonstrate a much higher frequency of careful building patterns (*Aa*, *AD*, *AA_*, *_AA*, *AA*). More specifically, students in Cluster 1 frequently increased the conceptual content of their programs in consecutive steps with a long amount of time spent on at least one of those steps (*Dd*, *dD*), spent a long time for increasing concepts in one steps and then took another long step decreasing concepts of their program (*DE*), or spent a long time at least on one step to increase conceptual content of their programs and not only failed in increasing the level of correctness, but also jumped back to the point where no test was passed (*JJ*, *Jj*). On the other hand, students in Cluster 2 did considerably less "tinkering" while focusing on large incremental building steps, in which they often spent a long time building their program. They often had long steps in which they added more concepts to the code and successfully increased its correctness (or at least did not degrade code correctness) (*AD*). They had these building steps more frequently when they started developing their program (*_AA*), while they were on mid-stages of code development (*AA*, *Aa*, *AD*), and also at the time they ended development (*AA_*).

We think that the behavior-based split separated the students into the groups that Perkins et al. (1986) called *tinkerers* and *movers* [19]. Movers gradually add concepts to the solution while increasing the correctness of the solution in each step. On the other hand, tinkerers try to solve a programming problem by writing some code and then making changes in the hopes of getting it to work.

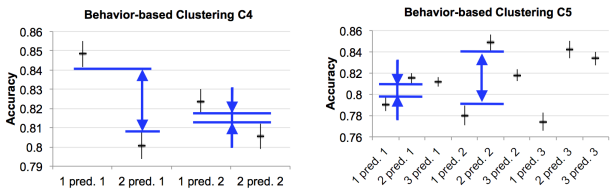
8 BEHAVIOR-BASED CLUSTERS VALIDATION

The bottom five rows under the header "Prediction diff." in Table 2 describe between-cluster model prediction differences, in terms of both the scores and the noteworthy clusters. None of the behavioral clustering approaches were scored as *ideal*, as we have not found at least two clusters that were voted as sufficiently different by the cluster models of student learning. However, all of the clustering approaches received an *expected* score: there was one cluster in each approach that dominated at least one other cluster.

Figure 3 graphically illustrates some of these results. The accuracy of each cluster's model cross-prediction for behavior-based clustering C4 is shown in Figure 3(a). There, we see that the Cluster 1 model wins when predicting test data from both clusters. Figure 3(b) is an illustration of cross-prediction accuracy differences in the case of Behavior-based clustering C5. Cluster 2, here, has superior prediction accuracy over Cluster 1 in both cases. In both of these figures, we see an *expected* case of one cluster model domination (as defined in Figure 1). We chose to visualize these particular clustering results since they represent two typical cases: C4 with two clusters only; and C4 with three clusters, where we only check two out of three prediction tasks to contrast Clusters 1 and 2.

9 ANALYSIS OF CLUSTER DIFFERENCES

The results of cross-prediction using behavior-based cluster models demonstrated that the discovered clusters (tinkerers, movers) were not performance-based stereotypes. In other words, the two clusters that we found did not differ sufficiently to form stereotypes that could better predict student performance and serve as a basis for personalization. While the clusters failed to separate students into



(a) (b)

Figure 3: Between-cluster student group model prediction accuracy differences for Behavior-based clustering approaches (a) C4 and (b) C5.

classic performance-based stereotypes (such as weak or strong), we observed that they separated students into distinctive groups with stable but different behaviors. Given the belief of some programming instructors that tinkering is not an ideal problem-solving behavior, we wanted to perform a deeper performance-focused analysis of our discovered behavior-based clusters. In this section, we inspect cluster performance in more detail.

Tinkerers Are Less Efficient and Have Lower Grades. To gain an insight into how the two behavior groups differed in terms of their performance, we looked into a set of performance measures that included: 1) the number of attempted problems; 2) the number of solved problems; 3) the average steps taken to solve the problem, where steps were only test, run, and submit snapshots; 4) the average time (in seconds) spent on solving the problem; 5) the effectiveness score; and 6) the final course grade. Effectiveness score is a measure of instructional efficiency and represents student performance on the problems that a student solved, as well as the mental effort that a student spent on solving those problems. Here, we chose the time on problem-solving as an approximate measure of student mental effort and compute an effectiveness score, as introduced in [17].

Table 6 presents performance statistics for each of the aforementioned measures among students in Cluster 1 (tinkerers) and students in Cluster 2 (movers) (note that these clusters were plotted in Figure 2). A Wilcoxon ranked sum test was performed to measure the difference on each performance measure in Cluster 1 and Cluster 2. As the table shows, there is a significant difference between the two clusters in several cases. On average, students in Cluster 2 took fewer steps to solve the problem ($M_2 = 3.4, M_1 = 5.9$), were faster at solving the problems ($M_2 = 630.0, M_1 = 998.1$), and as a result, were also more efficient in solving the problems ($M_2 = 0.4, M_1 = -0.3$). Furthermore, the average student grade was also higher in Cluster 2 than in Cluster 1 ($M_2 = 3.4, M_1 = 2.9$). While all parameters commonly used as signs of good performance point to Cluster 2, we should be careful when interpreting this result as a clear sign of the superior problem-solving abilities of students in Cluster 2. Making fewer larger steps is the very essence of problem-solving approach of Cluster 2, and it is no surprise that students from this cluster looked more efficient. On the other hand, there was no significant difference between clusters in respect of the number of solved problems, although students in Cluster 2 attempted more problems and solved more problems, on average, as compared to those in Cluster 1.

Performance Measure	*Cluster 1*	*Cluster 2*	*Wilcoxon Test*
#Attmpted Probs.	81.3 (1.7)	82.8 (1.3)	73,456
#Solved Probs.	68.1 (1.4)	70.9 (1.0)	68,860
Avg. Attempts to Solve	5.9 (0.2)	3.4 (0.1)	$123{,}790^{***}$
Avg. Time to Solve	998.1 (27.5)	630.0 (16.7)	$111{,}950^{***}$
Effectiveness Score	-0.3 (0.1)	0.4 (0.0)	7253^{*}
Course Grade	2.9 (0.2)	3.4 (0.1)	$43{,}068^{***}$

$^{*}: p < .05$; $^{***}: p < .001$

Table 6: Performance statistics (Mean,SE) for Cluster 1 (N = 295), and Cluster 2 (N = 503). Wilcoxon rank sum was performed to compare performance of Clusters 1 and 2.

Patterns Tend to Distinguish Low vs. High Performers. From what we observed, we know that one group was thinking in a constructive manner; that is, students in Cluster 2 often thought for a long time, added concepts, and increased code correctness (patterns *_AA*, *AA*, *AA_* in Figure 2). On the other hand, students in Cluster 1 seem to be weaker because they had more unsuccessful steps, they added concepts with no test being passed, or they changed (added/removed) concepts that did not influence the code's correctness (see patterns *Dd*, *dD*, *DE* in Figure 2). Apparently, Cluster 1 represents students who are less efficient in their problem-solving – evaluated by performance measures like effectiveness score, and average attempts for solving the problem. As a result, it seems likely that weaker students would be in this group.

When we investigated the relationship between the micro-patterns in each group and the performance measures further, we found that certain patterns are positively or negatively associated with performance[3]. In particular, some patterns that represent mostly tinkering behavior were negatively related to both the number of problems that student solved and their effectiveness score (*_jj*, *JjJ*, *ic_*, *_JJJ*, *jJJ*, *JJk*, *_JK*, *FF*). On the other hand, we found an instance of a constructive building pattern (*_AAD*) to be positively associated with both of these measures. Additionally, we observed that a pattern could have a different impact on different measures. In our case, pattern *bA* was positively related to the number of solved problems and was negatively related to effectiveness score.

Both Groups Include a Mixture of Strong and Weak Students. Why the tendency toward low- and high- performance among tinkerers and movers did not result in a grouping that accurately reflects performance-based stereotypes? This can be explained by elaborating on how weak and strong students were distributed across the two groups. There were both strong and weak students who exhibited similar problem-solving behavior. To check this hypothesis, we compared the overlap between the clustering that resulted in two groups of tinkerers and movers (i.e., clustering *C4*) and the two performance-based clustering (i.e., *Problem Solved*, and *Percent Correct*). We found little overlap between group labels that were found by these clustering approaches. This is sufficient evidence to let us conclude that there were both weak and strong students among movers and tinkerers.

It appeared that strong and weak students were dispersed within each behavior group. We observed that a large number of students in Cluster 1 (tinkerers) were strong students. These students performed well, but they exhibited the same problem-solving behavior as poor students. This clarifies that behavior-based clusters represented different behaviors in solving problems, rather than the classic weak/strong performance groups.

[3]Generalized linear model was used to model the performance measure of interest. Dependent variables were micro-patterns that had little correlation, if any.

10 DISCUSSION AND FUTURE WORK

We have set out to find at least two groups of students that could be considered to learn differently, as captured by the models of their learning. Just like in [28], where the domain is K-12 math, we found that, across all of the grouping/clustering approaches that we have considered, there is always a sub-sample of students who can effectively be used to build a model of learning for the whole population. While this is not the result we hoped for, it conveys an important message. This means that finding a useful learning-focused stereotype, like *good students* or *slow students*, is not trivial. There might be students who approach learning differently, but the distinction between these approaches are orthogonal to the conventional dimensions that we apply to quantify learning.

A set of simpler, as well as more advanced, behavior-based student grouping approaches that we tried did not result in discernible differences in cross-prediction accuracies. There is always one sub-population of students that contributes to a model that could be universally used for all. Our hypothesis is that adapting student models by swapping alternative parameterizations based on student stereotypes is not the correct approach. Although classic stereotype models have demonstrated their new value in the educational context as a basis for behavior prediction and personalized intervention, they seem to be failing as alternative models of student learning.

To date, the basis for our conclusion is limited: We looked at data from traditional courses and MOOCs in one field that originated from one University in Finland. The education system there might be different from the rest of the world and the sample of students could have influenced the behavior grouping and the performance of our models. In future work, we would like to obtain a larger, more representative sample of student data and re-run our analysis to validate and reconfirm our findings. Furthermore, we performed clustering separately on student demographic and performance data, and also on the genome data. In future work, it would be interesting to explore the clustering and student modeling using a combination of these sources. Finally, although the discovered clusters of tinkerers and movers were not useful for modeling student learning, they could be beneficial to researchers for other kinds of personalization. In particular, future work should investigate whether we can recognize a student as a tinkerer or mover sufficiently early and whether this early classification can be used to reduce less productive tinkering behavior using proper scaffolding.

ACKNOWLEDGMENTS

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF award OCI-1053575. Specifically, it used the Bridges system, which is supported by NSF award ACI-1445606, at the Pittsburgh Supercomputing Center.

REFERENCES

[1] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2014. Engaging with massive online courses. In *World Wide Web Conf.* 687–698.

[2] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. 2002. Sequential pattern mining using a bitmap representation. In *Knowledge Discovery and Data Mining Conf.* 429–435.

[3] Lori Breslow, David E Pritchard, Jennifer DeBoer, Glenda S Stump, Andrew D Ho, and Daniel T Seaton. 2013. Studying learning in the worldwide classroom: Research into edX's first MOOC. *Research & Practice in Assessment* 8 (2013), 13–25.

[4] Peter Brusilovsky and Eva Millán. 2007. User models for adaptive hypermedia and adaptive educational systems. In *The Adaptive Web: Methods and Strategies of Web Personalization*, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Neidl (Eds.). Lecture Notes in Computer Science, Vol. 4321. Springer-Verlag, Berlin Heidelberg New York, 3–53.

[5] John Champaign, Kimberly F Colvin, Alwina Liu, Colin Fredericks, Daniel Seaton, and David E Pritchard. 2014. Correlating skill and improvement in 2 MOOCs with a student's time on tasks. In *Learning@scale Conf.* 11–20.

[6] Albert T. Corbett and John R. Anderson. 1995. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction* 4, 4 (1995), 253–278.

[7] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* 9 (2008), 1871–1874.

[8] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. 2016. The SPMF Open-Source Data Mining Library Version 2. In *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*. 36–40.

[9] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *Computer-Human Interaction Transaction* 22, 2 (2015), 7.

[10] Julio Guerra, Shaghayegh Sahebi, Yu-Ru Lin, and Peter Brusilovsky. 2014. The problem solving genome: Analyzing sequential patterns of student work with parameterized exercises. In *Educational Data Mining Conf.* 153–160.

[11] Roya Hosseini and Peter Brusilovsky. 2013. JavaParser: A Fine-Grained Concept Indexing Tool for Java Problems. In *AI-supported Education for Computer Science Workshop*. 60–63.

[12] Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Exploring Problem Solving Paths in a Java Programming Course. In *Psychology of Programming Interest Group Conf.* 65–76.

[13] Jonathan Huang, Chris Piech, Andy Nguyen, and Leonidas Guibas. 2013. Syntactic and functional variability of a million code submissions in a machine learning mooc. In *AIED 2013 Workshops Proceedings Volume*. 25–32.

[14] Judy Kay. 1994. Lies, damned lies and stereotypes: pragmatic approximations of users. In *Fourth International Conference on User Modeling*, Alfred Kobsa and Diane Litman (Eds.). MITRE, 175–184.

[15] René F Kizilcec, Chris Piech, and Emily Schneider. 2013. Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the third international conference on learning analytics and knowledge*. ACM, 170–179.

[16] Christian Murphy, Gail Kaiser, Kristin Loveland, and Sahar Hasan. 2009. Retina: helping students and instructors based on observed programming activities. *ACM SIGCSE Bulletin* 41, 1 (2009), 178–182.

[17] Fred GWC Paas and Jeroen JG Van Merriënboer. 1993. The efficiency of instructional conditions: An approach to combine mental effort and performance measures. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 35, 4 (1993), 737–743.

[18] Philip I Pavlik Jr, Hao Cen, and Kenneth R Koedinger. 2009. Performance Factors Analysis – A New Alternative to Knowledge Tracing. In *Artificial Intelligence in Education Conf.* 121–130.

[19] David N. Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.

[20] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *ACM Technical Symposium on Computer Science Education*. 153–160.

[21] Elaine A. Rich. 1983. Users are individuals: individualizing user models. *International Journal on the Man-Machine Studies* 18 (1983), 199–214.

[22] Kelly Rivers and Kenneth R Koedinger. 2013. Automatic generation of programming feedback: A data-driven approach. In *AI-supported Education for Computer Science Workshop*, Vol. 50–59.

[23] Kshitij Sharma, Patrick Jermann, and Pierre Dillenbourg. 2015. Identifying Styles and Paths toward Success in MOOCs. In *Educational Data Mining Conf.* 408–411.

[24] Krisztina Tóth, Heiko Rölke, Samuel Greiff, and Sascha Wüstenberg. 2014. Discovering Students' Complex Problem Solving Strategies in Educational Assessment. In *Educational Data Mining Conf.* 225–228.

[25] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding Students' Learning Using Test My Code. In *Innovation and Technology in Computer Science Education*. 117–122.

[26] Xu Wang, Diyi Yang, Miaomiao Wen, Kenneth Koedinger, and Carolyn P Rosé. 2015. Investigating How Student's Cognitive Behavior in MOOC Discussion Forums Affect Learning Gains.. In *Educational Data Mining Conf.* 226–233.

[27] Miaomiao Wen and Carolyn Penstein Rosé. 2014. Identifying latent study habits by mining learner behavior patterns in massive open online courses. In *Conf. on Information and Knowledge Management*. 1983–1986.

[28] Michael Yudelson, Steve Fancsali, Steve Ritter, Susan Berman, Tristan Nixon, and Ambarish Joshi. 2014. Better Data Beats Big Data. In *Educational Data Mining Conf.* 205–208.

[29] Michael Yudelson, Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Investigating automated student modeling in a Java MOOC. In *Educational Data Mining Conf.* 261–264.