Educational Technology Quarterly, Vol. **2023**, Iss. 4, pp. 458-480 https://doi.org/10.55056/etq.611

Mining student coding behaviors in a programming MOOC: there are no actionable learner stereotypes

Muskan Yadav

Northumbria University, Sutherland Building, Northumberland Road, Newcastle-upon-Tyne, NE1 8ST, England, United Kingdom

Abstract. Education often involves categorizing students into broad groups based on perceived attributes like academic abilities, learning pace, and unique challenges. However, the validity and applicability of these stereotypes require closer examination. This research investigates student grouping factors, exploring both conventional variables like gender and education level, as well as innovative methodologies that utilize students' problem-solving behaviors. The study critically evaluates the effectiveness of these grouping techniques in capturing and distinguishing students' diverse learning patterns. Through a comprehensive analysis of ten methodologies used to cluster students in traditional programming courses and programming MOOCs, we aim to reveal how students from different cohorts exhibit varying learning behaviors and outcomes. By examining diverse models of student learning, we assess whether students in distinct groups indeed demonstrate discernible disparities in their educational journeys. Our meticulous data analysis uncovers compelling insights that challenge the notion of predefined student stereotypes and their practical utility within group-based adaptation settings. This research contributes to the discourse on student grouping by highlighting the limitations of traditional categorizations and introducing innovative approaches to understanding student diversity and tailoring educational interventions accordingly. By transcending simplistic generalizations, we strive to foster a nuanced understanding of students' individual strengths, challenges, and potentials, promoting inclusive and effective educational practices.

Keywords: individual differences, Java, MOOC, student modeling

1. Introduction

It has been observed that student performance in a MOOC could vary considerably with many students solving only a fraction of problems or dropping from the MOOC completely [3]. Consequently, considerable research also focused on explaining and predicting student performance in MOOC. It has been argued that the ability to predict performance could help to identify a cohort of weaker students sufficiently early to help them in succeeding a MOOC. Early attempts to identify weak and strong students focused mostly on examining various demographic features such as gender, level of education, or country of origin as a source for performance prediction [3]. Researchers would examine classification approaches employed to categorize the diversity of studies in educational data mining and review the research problems addressed using these methods [2]. Studies also explored the challenges and opportunities in the development of effective Social Learning Analytics (SLA), highlighting the turbulent

muskantesol@gmail.com (M. Yadav)
https://uk.linkedin.com/in/muskan-yadav-86990b206 (M. Yadav)
0009-0006-4427-5366 (M. Yadav)

© Copyright for this paper by its authors, published by Academy of Cognitive and Natural Sciences (ACNS). This is an Open Access article distributed under the terms of the Creative Commons License Attribution 4.0 International (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

nature of learning landscapes driven by technology [7]. These attempts brought little success, on the contrary, a significant number of both strong and weak students were found within each demographic category. The more recent research attempted to identify cohorts of strong and weak students by examining student course behavior through MOOC log mining. This research focused mostly on coarse-grain traces of student access to various parts of MOOC such as videos, questions, or discussion forums. While many attempts focused on behavior mining brought interesting results [23, 25, 26], the results of different studies were to some extent contradictory. In some cases more diligent access to course content was a positive sign of good performance [22]. In other cases, it was negatively correlated with success [4], or skipping content was a good sign [11]. It has been argued that patterns of student access to course content are, to a considerable extent, influenced by the starting level of knowledge rather than by differences in student approaches to learning.

The relevance of Massive Open Online Courses (MOOCs) in contemporary education cannot be overstated. MOOCs have revolutionized the way individuals access and engage with educational content. In an era marked by the democratization of knowledge and the increasing demand for lifelong learning, MOOCs offer a flexible and accessible avenue for learners worldwide. Their significance has been further magnified in recent times, as the global COVID-19 pandemic forced traditional educational institutions to explore online alternatives. MOOCs became instrumental in ensuring continuity in education during these challenging times, showcasing their adaptability and scalability.

Moreover, MOOCs have opened up educational opportunities to a diverse audience, transcending geographical boundaries and economic constraints. They provide a platform for learners to acquire new skills, explore different fields of study, and enhance their employability. The accessibility of MOOCs has made them particularly appealing to working professionals seeking to upskill or pivot to new career paths.

Despite their transformative potential, MOOCs also face challenges, including low completion rates and the need for effective strategies to support learner engagement and success [12]. Hence, understanding student performance and predicting outcomes in MOOCs remains a critical area of research.

Our research separates strong and weak learners by using a very different kind of behavior analysis. First, we focused on problem-solving behavior, which, we hoped, is less affected by the starting knowledge and more related to student individual learning approach than content access behavior. Second, we examined student behavior in a more finer-grain level tracing individual steps to solve each problem rather than registering all that as one content-level access. Finally, before moving to group elicitation and clustering, we ensured that we found reliable and repeatable patterns of behavior for every student. The result of our behavior mining attempt was, however, unexpected. While we were able to discover groups of students that considerably differ by their problem-solving behavior patterns, these groups were not the expected stereotypical "weak" and "strong" students and were useless for stereotype-based performance prediction. Instead, the differences between groups represented different problem-solving styles that have some correlation with performance, but, as our data demonstrated, does not define it.

In this paper, we offer a brief review of similar work including MOOC behavior mining and analysis of assessment data in programming. In two separate sections, we introduce our dataset and the performance prediction approach that we use to assess our ability to distill

performance-based stereotypes. To demonstrate some simple application of this approach, we start by confirming the absence of demography-based stereotypes. Following that, we present our sequence-based behavior mining approach and its application to our dataset to discover patterns of student problem-solving behavior. Next, we present and discuss the results of behavior-based clustering. We conclude with a general discussion of lessons learned.

1.1. Decoding programming assessment data

The scrutiny of programming problem solutions submitted by students as assignments has garnered considerable interest in recent years. In recent studies, the utilization of submission data has been instrumental in uncovering diverse approaches, both correct and incorrect, to tackle the same problem [9, 15]. Moreover, researchers have employed this data to construct an intelligent scaffolding system [21], model students' knowledge within the context of program development [20, 28], predict students' grades [16], and gain insights into students' coding behavior through conceptual analysis of program code [14].

By utilizing the assembly and delivery information pertaining to participants enrolled in a Java Massive Open Online Course (MOOC), this scholarly article brings a noteworthy value to the current repository of knowledge on the analysis of assessment information. This study encompasses three principal aims: (1) comprehending the distinct patterns characterizing individual problem-solving behaviors in coding, (2) evaluating the impact of these identified behaviors on students' performance within the programming course, and (3) investigating the ramifications of these behaviors for the accurate modeling of student knowledge.

2. Data

Sourced from a selection of four core programming study programs, which encompassed Massive Open Online Courses (MOOCs), this study acquired information from a research-oriented European university spanning the years 2014 and 2015. These courses primarily focused on introducing students to the programming concepts using the Java programming language. Each course spanned a duration of seven weeks and involved a series of programming assignments of varying complexity. To carry out their tasks, students utilized the NetBeans integrated development environment.

The programming tasks were administered using the Test My Code-plugin [24], a tool employed for automatic code assessment and data collection. Data from students who granted permission for this study were meticulously recorded, encompassing their keystrokes, assignment specifics, and unique student identifiers. The collected data was then aggregated and analyzed to capture noteworthy instances within the respondents' performance. Specifically, this study focused on key events such as running the program, executing program tests, and assessment. Furthermore, to facilitate a thorough examination of the transitional stages leading to significant events, the initial five general occurrences involving the insertion or deletion of text were also incorporated. Posthoc analysis employing JUnit test sets was conducted to extract insight. Furthermore, JavaParser [13] was employed to extract programming concepts associated with each problem-solving state.

Alongside the collection of programming process data, the subjects also took up an exercise, which inquired about their age, sex, programmer experience, and latest educational qualification. 2,739 unique students initially enrolled in the courses. 1,788 students formed the initial sample (with a cutoff at 2,500 recorded events, representing approximately the 33rd percentile of the workload during the first week of the course). The number of students who completed the information survey was 798, and they constitute the final batch we analysed. Table 1 provides essential statistical information regarding all the participants.

Table 1
Demographic profile of the students who participated in the courses.

Course	Student sample		Age			Gender			Programming background			Prior education		
	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%

The timeframe, spanning the years 2014 and 2015, may be considered somewhat dated in contemporary times, but its relevance persists in programming and educational research. The study's focus on introducing students to Java programming and fundamental concepts remains pertinent, as core programming principles endure amidst technological evolution. Moreover, insights into Massive Open Online Courses (MOOCs) engagement from this era offer valuable lessons in the face of the increased reliance on online education today. While specific tools like NetBeans and code assessment plugins may have evolved, the overarching idea of using integrated development environments (IDEs) and automated code assessment tools remains relevant for designing modern programming courses. The collection and analysis of programming process data, including keystrokes and programming concepts, continue to inform improvements in programming education and tools. Demographic data concerning age, gender, experience, and qualifications, 'which can be seen in table 1 still influences learner performance and engagement, offering insights into modern-day learners. Finally, the study's longitudinal approach, tracking students over time, provides enduring value in understanding learning patterns and the long-term impact of programming education.

3. The assessment approach

In this study, our objective is to investigate the effectiveness of segregating students into various cohorts for the purpose of adaptation per group. Specifically, our aim is to determine whether distinct groups of students can be identified, exhibiting diverse learning patterns (such as traditionally labeled "high-performing" and "low-performing" students). To assess the achievement of our desired "adaptation-level" division across multiple grouping strategies, we employ variations in models depicting student learning within each group as a criterion. We establish a guiding principle that distinct differences in learning approaches among groups should be reflected in the performance-based variations observed within their respective models.

Prior to exploring innovative methods of segregating students based on how they rank as a programmer, our examination observed relatively easy strategies for grouping. These simpler approaches involve pre-existing knowledge as well as post hoc information. We compared the advanced techniques with the inferior ones and assessed the distinctiveness of our behavior-based approach. Once the evaluation of the simpler grouping methods is completed, we will revolve around the exploration of student clustering based on what we mined on programmer behavior. To validate the efficacy of all the methods employed, we will utilize learning and prediction models.

4. Using models of student learning for grouping/cluster validation

4.1. Modeling student learning

Before discussing validation of student groupings/clusters, we would like to describe our student modeling approach. For modeling student knowledge acquisition, we considered models based on logistic regression. In particular, we used a model called Performance Factors Analysis [18]. Within this model, individual student abilities are represented as arbitrary pointers denoted by θ_i, while concept difficulties are shown as fixed-factor intercepts denoted by β_k. Moreover, the learning rates for skill acquisition in all submissions are represented by γ_k and ρ_k , respectively. The canonical form of the Probabilistic Factor Analysis (PFA) is expressed in equation (1). In relation to the current assessment, the variables s_{ik} and f_{ik} denote the individual counts of accomplished and unsuccessful attempts made on a particular concept beforehand.

$$\Pr(Y_i = 1 \mid \theta, \beta, \gamma, \rho) = inv.logit\left(\theta_i + \sum_k (\gamma_k s_{ik} + \rho_k f_{ik})\right) \tag{1}$$

The selection of our modeling approach was driven by the cumulative nature of PFA, wherein the combined influence of various elements within a student's work contributes to an aggregated signal that determines the overall success or failure. On the other hand, the contrasting modeling methodology known as Bayesian Knowledge Tracing (BKT) [5] is not specifically designed to address scenarios involving the presence of multiple interconnected concepts within student interactions. This limitation arises from the violation of the independence assumptions inherent in the Hidden Markov Model employed by BKT.

Two substantial modifications were implemented in the PFA model, leading to improved overall model fit. The first significant change involved transitioning from globally defined concepts applicable to all problems to a more refined approach using within-problem concepts. Indeed, a for-loop concept crucial for one problem could be irrelevant for another. Second, we used each concept's usage count within the problem code toward the opportunity count and $log1p$-transformed opportunity counts after that. The effectiveness of both these alterations was demonstrated in the study conducted by Yudelson et al. [28]. Their investigation involved analyzing data from the same data source but across different academic years. These modifications only changed the definitions of what is a concept and how the success and failure attempts are accumulated.

Unlike the study conducted by Yudelson et al. [28], we adopted a distinct approach to data preprocessing. Our approach involved several key differences. Firstly, we treated each snapshot of student code as an individual and self-contained unit of data. The success of a snapshot was determined by the passing of all associated tests, while unsuccessful attempts indicated when the code failed to pass all tests. Secondly, our exclusive focus was on snapshots where students intentionally tested, executed, or finalized their code, while disregarding intermediate snapshots for the purpose of student modeling. Thirdly, our analysis encompassed all the concepts employed in students' code, rather than solely examining modifications in concepts with or without specific considerations for removals, as described in [28]. We found that this approach, under our data preprocessing setup, yielded better model performance. Lastly, our analysis was restricted to a final sample size of 798 out of the initial 1788 students.

As a result of our modifications, the number of concept parameters in the PFA increased from $143 \times 3 - 1 = 428$ to $143 \times 240 \times 3 - 1 = 102{,}959$. Considering the sparsity of the problem-concept matrix, the parameter count was effectively reduced to 40,625 ($13{,}542 \times 3 - 1$). Moreover, given the substantial data volumes (approximately 392,000 student submissions), conventional statistical packages were not feasible for analysis. To address this, we employed a customized version of the LIBLINEAR tool [6]. The modification, available at [1], introduced an additional solver capable of utilizing grouped L^2-penalties to approximate random factors. This modified version of LIBLINEAR retained the original tool's ability to effectively handle large datasets.

4.2. Validating by cluster models' cross-prediction

We sought a minimum of two distinct cohorts within our sample of students who demonstrate contrasting learning patterns. Drawing inspiration from [27], we adopt a specific methodology. Initially, we divide the relevant students among n number of clusters. Subsequently, we conduct 20 samplings within every group, extracting 80 students for training purposes and reserving an additional 20 students for testing. From each training set comprising 80 students, we construct a unique subsample model. Next, we utilize the $20n$ models and forecast three sub samples: one aligning with the group for which the model was developed, and the remaining two originating from the other group(s). Ultimately, we plot n^2 model accuracies, capturing both mean values and standard errors. Out of these accuracies, n reflect model performance within each specific group, while the remaining accuracies represent performance comparisons between the groups.

We categorize individuals into separate cohorts, characterized by unique patterns of knowledge acquisition, is predicated on the discernible differentiation between the performance of within-cohort models and across-cohort models. Specifically, we expect intra-group models to surpass that of inter-group. To exemplify this criterion, consider the depiction in figure 1, with the "ideal" separation highlighted in vivid red. In this scenario, the model developed on group A exhibits superior performance compared to that of group B when making predictions based on insights from group A. This clear delineation between groups A and B indicates that their respective models, when cross-predicting, exhibit subpar performance beyond their original source student sub-samples. An "expected" case, symbolized by the bold blue color, recurrently observed in [27], demonstrates one model's dominance over the others regardless of the test

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

data sub-sample. Instances of this nature are illustrated by the blue line, representing model *B* versus model *C*. Lastly, a "sub-optimal" scenario, denoted by the vibrant green color, materializes when one model prevails on its "own ground" (model *A*) but fails to outperform or succumbs to the other model (model *C*).

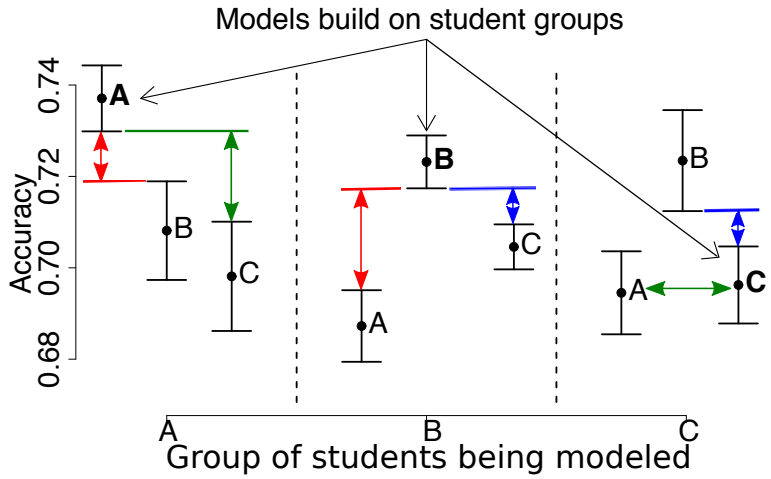

Figure 1: Differences accuracy between the groups are represented by the mean values and standard errors. An *ideal* scenario is denoted by red arrows, while a *expected* case is indicated by blue arrows. Similarly, a *sub-optimal* situation is highlighted with green arrows.

5. Simple student grouping

To demonstrate the practical implementation of our assessment approach, we initially explore more straightforward methods of student grouping based on demographics and course performance. This data is conveniently accessible to us through background information collected throughout the course or by utilizing overall course statistics available at the conclusion of the study. The summarized groupings resulting from these approaches are outlined in the first five rows of table 2.

Gender. Approximately 71% of the student population consisted of males.

Education level (*Edu.*) resulted in three distinct groups. Among the students, 524 individuals had completed Primary and Secondary Education, while 154 students pursued higher education at the college level. Additionally, 120 students were enrolled in graduate-level programs.

Transaction volume (*#Trans.*). The groups formed were low, medium, and high, considering the cumulative sum of attempts made at solving problems. An investigation conducted by the authors in [27] employed a comparable methodology to examine student groupings. Their findings indicated that a subset of students who contributed a more extensive volume of data and covered a wider range of topics exhibited a notably superior global model.

Problems Solved (*P.Solv.*). The results were generated using an agglomerative approach, utilizing Euclidean distance, applied to four course-level count variables: problems resolved (with at least one submission achieving perfect correctness), partially resolved problems (with at least

Table 2
Validating approaches to clustering students.

		Group/cluster label overlaps									Prediction diff.	
Approach	*Cluster sizes*	*Edu.*	*#Trans.*	*P.Solv.*	*%Corr.*	*B1*	*B2*	*B3*	*B4*	*B5*	*Score*	*Cluster to note*
Gender†	570, **228**	0.36	0.03	0.06	0.07	0.10	0.02	0.13	0.18	0.19	0.33	F.
Edu.‡	524, 154, 120		0.04	0.12	0.06	0.18	0.15	0.12	0.24	0.12	0.00	
#Trans.‡	266, 266, **266**			0.40	0.30	0.18	0.21	0.34	0.21	0.24	0.67	Hi
P.Solv.‡	218, 316, 264				0.09	0.15	0.18	0.31	0.09	0.04	0.00	
%Corr.‡	266, 266, 266					0.22	0.18	0.12	0.28	0.28	0.00	
B1	**383**, 415						0.58	0.43	0.58	0.51	0.67	1
B2	**416**, 382							0.70	0.44	0.51	0.67	1
B3	258, **158**, 382								0.40	0.51	0.67	2
B4	**295**, 503									0.69	0.67	1
B5	389, **272**, 137										0.67	2

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

one submission scoring above 0% correctness), attempted but unresolved problems (with at least one submission scoring 0% correctness), and unattempted problems. This grouping analysis produced distinct student groups: low (predominantly characterized by minimal problem attempts), high (mostly demonstrating successful problem-solving), and medium (encompassing all other students). This grouping scheme serves as an overarching measure of student performance.

Percent Correct (*%Corr*). The procedure categorized students into percentile-based groups (low, medium, and high) determined by their overall percentage of correctness during intentional code testing, execution, or submission. This classification differentiated students based on their level of conscientiousness.

5.1. Validating simple groupings

Top left quadrant of the "Group/cluster label overlaps" columns in table 2 contains pairwise similarities of five simple groupings. The similarities are computed as the largest overlap between the breakdowns of the students into the clusters over the total of 798 students. The overlaps are scaled to assume values from 0 to 1. There is no general threshold that we are aware of to be comparing these similarities against. We used ≤ 0.40 as an ad-hoc rule of thumb. Thus, all pairwise overlaps of simple groupings are small enough to be noteworthy.

We prioritize the group separation for cross-prediction in the following order: *ideal*, *expected*, and *suboptimal*. To simplify the assessment process, we assign scores to the separation levels seen in a group/cluster. A score of 1.0 indicates an *ideal* separation, a score of 0.67 suggests an *expected* separation, a score of 0.33 signifies a *suboptimal* separation, and a score of 0.00 indicates no significant separation. The discrepancies when making predictions are captured by "Prediction diff." columns of table 2, specifically in the top five rows. Amidst the five methodologies explored, solely two manifest non-zero ratings. Concerning gender distinctions, the model fashioned for female learners surpasses the model tailored for male learners in prognosticating test data pertaining to females, while both models exhibit commensurate performance in foretelling test

data concerning males. The model associated with the cluster featuring the highest student contribution exhibits superior performance compared to the other models, regardless of the test data being predicted.

6. Behavior-mining for clustering students

The fundamental concept underlying our behavior mining approach involves capturing and characterizing student problem- solving behavior at a micro-pattern level. These micro-patterns serve to elucidate the progression of students from incorrect solutions to the correct solution, as well as the growth of their knowledge across successive assignments. Our approach encompasses three distinct parts, each described in detail below.

In Part A, we initiated the process by analyzing and categorizing the intermediate programming steps taken by students, thereby capturing their programming behavior at each step.

In the subsequent stage, denoted as Part B, we utilized methodologies in consecutive motif extraction to identify consecutive micro-motifs from the amassed data. These micro-patterns represent the recurring patterns of actions undertaken by students as they navigate through the problem-solving process.

Building upon the extracted micro-patterns, Part C involved constructing a profile vector, or what we refer to as a "genome". This genome includes frequently occurring micro-patterns and summarizes an individual's ability to solve problems.

In order to ascertain the strength and dependability of our methodology in extracting problem-solving behaviors, we undertook a comprehensive assessment of the consistency of the behavior vector obtained from the micro-patterns. This evaluation was conducted to ensure the robustness and reliability of our approach in identifying and analyzing problem-solving behaviors.

6.1. Part A: Analyzing intermediary programming phases

In our quest to understand student problem-solving behavior, our initial focus was on investigating the trajectory of students as they tackled coding challenges. To gain insights, we employed a collection of intermediate programming steps known as "snapshots", which were captured during students' coding activities. These snapshots meticulously documented the code submissions and their corresponding correctness by evaluating them against a comprehensive suite of problem-specific tests.

Figure 2 depicts a pair of sequential snapshots representing a student's progress in solving the problem titled "Bigger Number". This particular problem required the student to develop an application capable of taking a couple of inputs as numerals before generating an output representing the bigger numeral. Student's code was subjected to three tests to evaluate its correctness. Test 1 and Test 2 verified the accuracy of the output when the first or second number was smaller, respectively. Test 3 aimed to ensure that the student did not include any unnecessary information in their program's output.

In the presented example, the initial snapshot (figure 2) showcased the student's initial attempt, which successfully passed Test 1 and Test 2 for cases when the first or second number was smaller, respectively. However, it failed Test 3 as it generated additional output when the second number was smaller. Following feedback on this issue, the student made improvements by

```java
import java.util.Scanner;
public class BiggerNumber {
    public static void main(String[] args) {
         Scanner input = new Scanner(System.in);
         System.out.println("Type a number:  ");
         int firstNumber = Integer.parseInt(input.nextLine());
         System.out.println("Type another number:  ");
         int secondNumber = Integer.parseInt(input.nextLine());
         if (firstNumber > secondNumber)
             System.out.println("The bigger number was: " + firstNumber);
         if (firstNumber < secondNumber)
             System.out.println("The bigger number was: " + secondNumber);
         else
             System.out.println("Numbers were equal: ");
    }
}
```

(a)

```java
import java.util.Scanner;
public class BiggerNumber {
    public static void main(String[] args) {
         Scanner input = new Scanner(System.in);
         System.out.println("Type a number:  ");
         int firstNumber = Integer.parseInt(input.nextLine());
         System.out.println("Type another number:  ");
         int secondNumber = Integer.parseInt(input.nextLine());
         if (firstNumber > secondNumber)
             System.out.println("The bigger number was: " + firstNumber);
        else if (firstNumber < secondNumber)
             System.out.println("The bigger number was: " + secondNumber);
         else
             System.out.println("Numbers were equal: ");
    }
}
```

(b)

Figure 2: *Bigger Number* program: (a) 1^{st} snapshot, (b) 2^{nd} snapshot.

incorporating an "else if" statement in their code (figure 2). Consequently, the modified program successfully passed Test 3, demonstrating the student's ability to avoid printing unnecessary information when there was a difference between the numbers.

Keeping up with Hosseini, Vihavainen and Brusilovsky [14], our approach to extracting programming behavior involves an initial examination of conceptual disparities between consecutive snapshots. This entails observing the inclusion or exclusion of specific concepts in each step and analyzing the correlation between these changes and the improvement or deterioration of program correctness. The quantification of conceptual discrepancies between two snapshots is determined by calculating the variance in concept counts. To mine these behaviors comprehensively, our methodology encompasses two primary stages: (a) the categorization of student

snapshot sequences within each problem, and (b) the identification of frequent behavioral micro-patterns (referred to as "genes") through sequence mining techniques applied to the labeled snapshots.

In the process of labeling the sequence of snapshots for each student in a given problem, the captured snapshots pertaining to that student stod collected then arranged chronologically. Labels were assigned to each snapshot in the sequence, taking into account the alterations made to the *programming concepts* and the *degree of correctness* in relation to the previous one. Coming to the sequence's initial snapshots, the corresponding prior snapshot for each one, sequentially speaking, is tagged as $\emptyset$, indicating no concepts and no passed tests. The labels employed for snapshot categorization are detailed in table 3.

Table 3
Programming style: snapshot-labeling.

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

To provide a practical example, let us consider the first snapshot in figure 2. Upon closer inspection, it is evident that the student has incorporated supplementary elements, resulting in an increased ratio of successful tests to 0.67 (passing two out of three tests). Consequently, the first snapshot is assigned the label "a". Similarly, for the second snapshot depicted in figure 2, the same label is also assigned. This determination is based on the student's inclusion of an additional concept ("if else") and a perfect ratio of passed tests (passing all three tests). As the Bigger Number problem only encompasses two snapshots for this particular student, the chain is tagged "_aa_." The "_" symbol denotes the beginning and end points of the sequence.

To enhance the distinction of brief steps from extended ones, an additional aspect can be incorporated into each label, signifying the duration of time invested in a snapshot. This aspect holds significance as it sheds light on the problem-solving behavior. To account for variations in programming speed among different students, individualized thresholds are employed to classify the duration dedicated to a stage or phase as brief or extended. Consequently, the labeling system encompasses characters in lower-case i.e. a to l. This denote *small* steps. Corresponding upper-case letters i.e. A to L signify *extended* actions. Let us consider the student presented in figure 2. 10 minutes is the median time distribution. 15 minutes went on code development in the initial snapshot. A tiny modification in the subsequent snapshot took an extra 2 minutes. The resulting label for their sequence of snapshots would be "_Aa_".

Overall, our labeling process encompassed a total of 137.504 snapshot sequence labels, derived from 1788 students' attempts. There were 241 distinct exercises. The value of these sequences' steps exhibited variation, spanning from 1 to 475, with an average size of 5.3. Specifically, 92.549 chains consisted of more than one step, 64.328 sequences consisted of more than two steps, 48.195 sequences consisted of more than three steps, and 38.768 sequences consisted of more

than four steps.

6.2. Part B: Extracting subtle patterns when solving problems

To extract frequent recurring styles in chains of problems solved, we utilized the SPAM algorithm [1], employing the implementation technique described in [8]. The information provided comprised 9254 sequences encompassing a minimum of two steps. Through the SPAM algorithm, we discovered trends manifesting throughout the *minsup* pertaining to our respondents. Our tactic was to use a modest threshold applicable to less common trends that may exclusively emerge within smaller student groups. Additionally, we enforced the absence of gaps during the SPAM analysis to ensure that the identified patterns consisted of contiguous steps. Lastly, the patterns were limited to a couple of steps in general to enable better tracking of the students' coding skill progression.

Refer to the snapshot[2]. Table 4 presents 20 of the most prevalent trends along with corresponding frequencies of happening.

Table 4
Frequently occurring programming patterns in students' problem-solving sequences: top 20 patterns.

Rank	*Pattern*	*Frequency*
1	AA	19.78%
2	AD	13.75%
3	_AA	12.17%
4	Aa	9.75%
5	AA_	8.69%
6	DD	8.53%
7	aA	8.31%
8	Ad	8.26%
9	Af	8.15%
10	JJ	7.22%
11	Ac	6.24%
12	DA	6.22%
13	_AD	6.18%
14	Dd	6.15%
15	JA	6.13%
16	Jc	6.04%
17	_Af	6.01%
18	dD	5.61%
19	DE	5.57%
20	Jj	5.45%

[2]Assuming an average sequence length of 5, the potential maximum number of patterns that could be discovered is on the order of 8: With 24 labels (a–l, A–L) available, there are 24^5 possible sequences obtainable, and the number of feasible substrings in a 5-character sequence amounts to $5 \times (5 + 1)/2 = 15$. Consequently, the overall count of detectable patterns from sequences of length 5 sums up to $24^5 \times 15 = 119{,}439{,}360$.

6.3. Part C: Leveraging micro patterns for behavior representation

We harnessed the micro-patterns extracted through sequential pattern mining to construct individual behavior profiles denoted as frequency vectors. These vectors captured the occurrence rates of each micro-pattern. To ensure uniformity along with comparability, we normalized frequency vectors, resulting in a cumulative frequency of 1 across the micro-patterns. This methodology draws inspiration from the seminal work presented in [10], where a similar approach was employed to discern relevant behaviors. Consistent with this earlier research, we refer to the discussed profiles as the *problem solving genome*.

Micro-pattern-based behavior vector's stability (referred to as the genome) and its ability to capture enduring characteristics of students, a method proposed in [10] was employed. This method involved splitting the student sequences into two halves and constructing behavior vectors from each half. By comparing the resulting half-vectors, it was possible to determine if the student's behavior profile exhibited stability. To assess this, the student sequences were split in two ways: random-split, where sequences were randomized across two splits, and temporal, where sequences were ordered according to timing and sorted among halves. Behavior vectors were constructed each and every one of the halves, after which pair-wise gaps were calculated. Jensen-Shannon (JS) measure, commonly employed for computing distances between frequency distributions, was used to calculate the distance between the half-vectors.

We utilized the Wilcoxon rank test and compared the distances like self and others. Significantly lower distance from self (*Mean* = 0.3485, *SE* = 0.0025) was observed in the random-split scenario (*Mean* = 0.6586, *SE* = 0.0010), with a p-value less than 0.0001. Talking about the temporal cases, distance from self in the split halves (*Mean* = 0.4249, *SE* = 0.0022) exceeded that of the random-split ones, albeit significantly less in quantity (*Mean* = 0.6534, *SE* = 0.0010), with the value of p less than 0.0001. These compelling findings offer robust support for the reliability of frequency considered micro in its pattern as a representation of students' behavior when it comes to tackling problems. Moreover, the behavior profiles derived from this approach showcase a distinctive capability to accurately depict individual students' ability to solve problems.

After establishing the stable profiles of student behavior through vector-based representation, our subsequent objective is to employ the micro-patterns as a means to categorize students according to their problem-solving styles. The ensuing section provides an elucidation of the identified behavior groups and their influence on student performance.

7. Building and examining behavior-based groups

7.1. Grouping into clusters based on similar behavior

The behavior vector of micro-pattern frequency was constructed for each student, and the students were clustered based on these vectors. The behavior vector was built using the students' distinct abilities on solving problems. The process encompassed the consecutive snapshots captured during their program development for various problems throughout the course. Five distinct settings were explored for clustering the students' behavior, as outlined in table 5. In the latter four settings, the snapshots were labeled based on concepts, correctness, and the duration spent on a respective snapshot, while the first setting did not consider time. Furthermore, there

was variation in the micro patterns utilized in the labeling process. Initially, 45 patterns were drawn from a 5% threshold applicable to, derived by setting the SPAM's *minsup* threshold to 5%. Subsequently, the settings utilized 245 patterns where a 1% threshold value was set on *minsup*.

Table 5
Settings of the frequent programming pattern clustering.

No.	*#Patterns*	*Minsup*	*Clustering method*	*#Clusters*	*Time*
1	45	5%	Hierarchical	2	
2	245	1%	Hierarchical	2	✓
3	245	1%	Hierarchical	3	✓
4	245	1%	Spectral	2	✓
5	245	1%	Spectral	3	✓

7.2. Interpreting discovered clusters

We embark on an in-depth exploration of behavior-based grouping, aiming to provide a more comprehensive understanding of its nuances. Our focus centers on settings utilizing two clusters to facilitate a clearer differentiation. After conducting a thorough analysis using both clustering similarity assessment and manual examination, we identified that the three configurations with two clusters successfully divided all students into either of two groups. The first group exhibited a proactive approach, characterized by a gradual progression of building steps. In contrast, the other group showcased a propensity for code manipulation without improving its correctness, often struggling through consecutive steps without achieving favorable outcomes. In the context of the three-cluster settings, we observed a comparable grouping of students, with an additional cluster capturing mixed behaviors that showed similarities to the other two clusters in specific subsets of micro-level characteristics. The classification of behaviors into distinct clusters (refer to table 5, row 4) obtained from the Spectral approach is visually depicted in figure 3. Carefully calibrated based on the divergence between the two clusters, the vertical axis of the graph showcases the frequencies of the 20 most prominent micro-patterns within both of them.

In the figure, at opposite sides of the plot, it illustrates distinct trends of micropattern frequencies. Cluster 1, on the left, has a greater "tinkering" occurrence (*Dd*, *dD*, *JJ*, *DE*, *Jj*). On the opposite side, cluster numbered 2 exhibits substantially increased frequency of meticulous building (*Aa*, *AD*, *AA_*, *_AA*, *AA*). Within cluster 1, students frequently partake in a series of iterative stages aimed at enriching the substantive aspects of their programs. This iterative process involves dedicating substantial time and effort to the stages such as *Dd* or *dD*, demonstrating their commitment to refining the content and structure of their programs. Moreover, they demonstrate instances where prolonged periods are spent on increasing program concepts in one step, followed by subsequent steps that involve a reduction in program concepts (*DE*). Another behavior observed in this cluster is investing considerable time in a step to augment program concepts, yet failing to improve correctness and sometimes regressing to a point where no tests are successfully passed (*JJ*, *Jj*).

In contrast, students in cluster 2 exhibit a significantly lower inclination toward "tinkering"

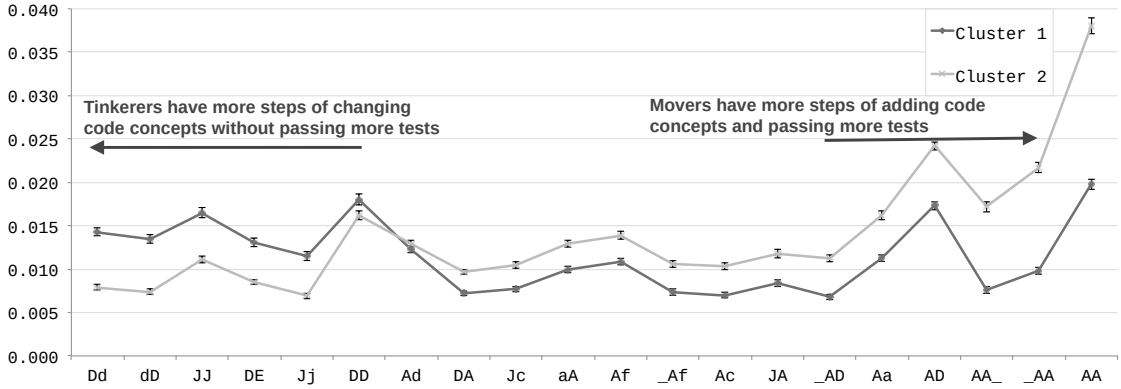

Figure 3: Top 20 programming patterns and their ratio of occurrence in each cluster. Patterns are ordered by absolute difference of ratios between Cluster 1 (tinkerers) and Cluster 2 (movers), and error bars represents standard error of the mean.

behaviors and instead prioritize large incremental building steps, devoting considerable time to program development. These students frequently engage in lengthy steps that involve incorporating additional concepts into the code, resulting in successful enhancement of correctness or, at the very least, the preservation of existing correctness levels (*AD*). These building steps occur more frequently at the beginning of code development (*_AA*) or even for steps in the middle (*AA*, *Aa*, *AD*). Stages towards the completion of their code (*AA_*) are also considered.

Upon conducting a thorough manual examination of the clusters, we observed that the division based on behavior patterns did not align with differences in students' performance. Instead, the clusters appeared to segregate students based on their preferred approach to problem-solving. These two groups can be mapped as *tinkerers* and *movers* [19]. The behavior of movers is characterized by a gradual addition of concepts to the solution, accompanied by a simultaneous increase in correctness with each step. Conversely, tinkerers adopt a different approach to solving programming problems. They initially write code and then proceed to make subsequent modifications in an attempt to achieve the desired outcome.

8. Validating behavior-based clusters

8.1. Overlaps between clustering approaches

Values in the top right quadrant of the "Groups/cluster label overlaps" rows in table 2 (rows that show overlaps with simplified groupings) are all ≤ 0.40 (below our ad-hoc threshold). This means that none of the five behavior-based clustering results align with demographics, background, or overall course performance. This is the proof that behavior-based clusters are orthogonal to simpler ways to group students and we consider that as a positive validation. Overlaps between behavior-based clustering results are all above 0.40, highest when 2 and 3 cluster versions of the same approach are compared (0.70 for B2 vs. B3 and 0.69 for B4 vs. B5). However, as clustering approaches B1 to B5 could be thought of as a sequence of incremental improvements, higher overlaps are expected.

8.2. Validating behavior-based clusters by cross-prediction

The "Prediction diff." section in table 2, specifically the five rows beneath others, provides insights into the between-cluster model prediction differences. Not a single approach achieved an *ideal* score, our analysis indicates that the observed clusters were not markedly different in terms of their learning approaches. Nevertheless, each clustering method yielded an *expected* output, as each approach identified one dominant cluster that outperformed at least one other cluster in terms of model predictions.

To graphically illustrate some of these results, refer to figure 4, where accuracies of cluster models cross-prediction are shown for Behavior-based clustering B1. In the given analysis, it is evident that cluster 1 emerges as the superior model when making predictions on test data from both clusters. To visually depict the accuracy differences in cross-prediction, figure 5 presents a graphical representation for Behavior-based clustering B5. Notably, in this case, cluster 2 exhibits higher prediction accuracy compared to cluster 1 across all test data partitions, as does cluster 3.

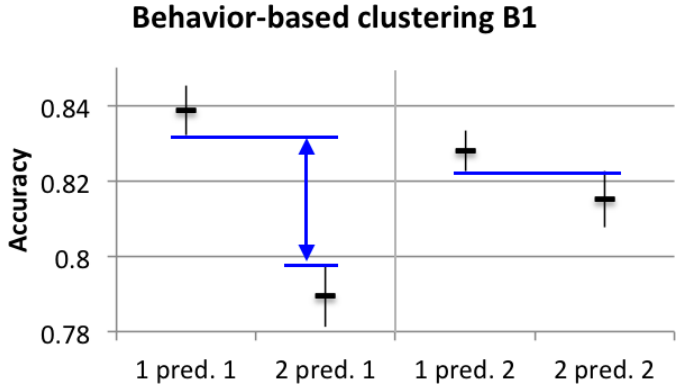

Figure 4: Capturing the essence of Behavior-based clustering B1: Unveiling the remarkable variations in prediction accuracy within student groups across different clusters.

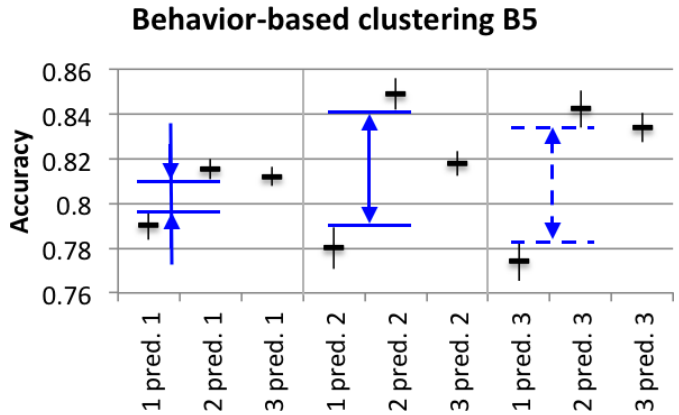

Figure 5: Presenting the divergent model prediction accuracies among student groups across clusters in Behavior-based clustering B5..

9. A deeper analysis of cluster differences

9.1. What differences do behavior-based clusters show?

Our investigation into performance prediction using behavior-based clusters yielded compelling outcomes, indicating that the clusters we uncovered (innovators, adventurers) deviated significantly from preconceived performance-based stereotypes. Put simply, the two clusters we identified failed to exhibit substantial performance disparities, which would have enabled the formation of stereotypes for predicting students' performance and facilitating personalized approaches. While the clusters did not effectively classify students into conventional performance-based categories (weak, strong), we did observe the formation of distinct groups with distinct and consistent behavioral patterns. Nonetheless, recognizing the prevailing belief among certain programming instructors that tinkering does not align with effective problem-solving behavior, we are now compelled to undertake a more profound analysis, focusing explicitly on performance within our identified pattern-based clusters. In this section, we aim to thoroughly scrutinize the performance aspects inherent to these clusters.

1. *Exploring the inefficiency and lower academic performance of tinkerers.*

To gain insights into the contrasting performance characteristics between the two behavior cohorts, we conducted an in-depth analysis of various performance indicators. These measures encompassed: 1) the volume of problem attempts undertaken by students, 2) the count of problems successfully solved by students, 3) the average number of procedural steps employed to arrive at solutions, 4) average duration devoted to problem-solving endeavors, 5) efficiency, and 6) overall grade of the completed coursework. The efficiency grade serves as the metric for effectiveness of course instructions, encompassing students' performance on solved problems and the cognitive effort invested in tackling them. As a proxy for mental exertion, we selected problem-solving time and employed the effectiveness score calculation method as introduced in [17].

Table 6 shows the relevant metrics capturing student performance in clusters 1 and 2 (tinkerers and movers, respectively). The Wilcoxon method assessed the performance measure difference between the two clusters. It reveals significant disparities between the clusters across multiple aspects. On average, cluster 2 students exhibited a lower number of steps taken to solve problems ($M_1 = 5.9$, $M_2 = 3.4$), showcased increased abilities when solving problems ($M_1 = 998.1$, $M_2 = 630.0$), plus, consequently displayed higher efficiency in problem-solving ($M_1 = -0.3$, $M_2 = 0.4$). To summarize, cluster 2 generally surpassed cluster 1 on grades ($M_1 = 2.9$, $M_{12} = 3.4$). It is crucial to exercise caution when interpreting these results as a definitive indication of cluster 2's superiority in problem-solving. Regarding the level of problem-solving proficiency, no significant disparities were found in inter-cluster variations. On average, cluster 2 demonstrated greater number of problems attempted and successfully solved.

2. *Patterns indicate a proclivity towards a range of performance levels, from low to high.*

From our observations regarding the clustering arrangements, we can deduce that one particular group exhibited a constructive thinking approach. This group comprises students in cluster 2,

Table 6
Performance comparison (Mean, SE) for Clusters 1 (N = 295) and 2 (N = 503) students. Wilcoxon test used for performance evaluation.

Performance measure	*Cluster 1*	*Cluster 2*	*Wilcoxon score*
#Probs. attempted	81.3 (1.7)	82.8 (1.3)	73,456
#Solved Probs.	68.1 (1.4)	70.9 (1.0)	68,860
Avg. no. of attempts	5.9 (0.2)	3.4 (0.1)	123,790***
Avg. time	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***

* : $p < .05$; *** : $p < .001$

who frequently engaged in prolonged contemplation, incorporated additional concepts, and enhanced the accuracy of their code (as indicated in figure 3). Cluster 1's students problem-solving abilities were comparatively subdued. Unlike cluster 2, they encountered more unsuccessful steps, introduced concepts without passing tests, or made changes (additions/removals) that had no bearing on code accuracy (refer to figure 3 and note *Dd*, *dD*, *DE*). We see in cluster number 1 students who display lower levels of efficiency in their problem-solving endeavors, as evident from performance metrics such as the effectiveness score and average attempts to solve problems. Therefore, it is plausible to conclude that weaker students are more likely to be part of this cluster.

Upon delving deeper into the connection between micro-patterns within each group and the performance metrics, we unearthed intriguing associations, whereby certain patterns demonstrated either positive or negative correlations with performance measures[3]. Notably, several patterns predominantly indicative of tinkering behavior exhibited negative associations with problems solved in terms of overall number and the effectiveness in doing so (*_jj*, *ȷȷ*, *ic_*, *_ȷȷȷ*, *jȷȷ*, *ȷȷk*, *_ȷK*, *FF*). Conversely, we discovered that a constructive building pattern (*_AAD*) displayed a positive correlation with both aforementioned measures. Furthermore, it became evident that a pattern could exert varying impacts on different performance metrics. In our analysis, the *bA* pattern showcased a positive relationship with the number of problems solved, while concurrently exhibiting a negative correlation with the effectiveness score.

3. *Both clusters encompass a blend of students exhibiting varying levels of proficiency.*

The absence of performance-based stereotypes within the clusters of tinkerers and movers can be attributed to the dispersion of weak and strong students across both groups. To explore the hypothesis that students with different levels of proficiency might demonstrate similar problem-solving behaviors, we conducted a comparison between the clustering outcomes of tinkerers and movers (clustering B4) and the performance-based clustering approaches (*Problem Solved* and *Percent Correct*). Intriguingly, we discovered instances where students with varying proficiency levels exhibited overlapping patterns in their problem-solving behaviors. The findings revealed

[3]This analysis employed a generalized linear model to establish the relationship between the performance measure of interest and the micro-patterns, which exhibited minimal or no discernible correlation.

minimal convergence between the groups identified by these clustering methods (refer to table 2, rows 4-5 in column B4, with an overlap of less than 0.30). Based on this compelling evidence, we can conclude that both students with weaker and stronger performance lie within both clusters (movers and tinkerers).

Notably, within cluster 1 (tinkerers), a significant portion of students demonstrated commendable academic performance despite exhibiting problem-solving behaviors similar to those of underperforming students. This observation serves to reinforce the notion that behavior-based clusters encapsulate distinct problem-solving approaches rather than aligning with conventional categorizations of weak and strong performance groups.

10. Concluding discussion

10.1. There are not even two different student stereotypes

Our objective was to identify a minimum of two student groups, not necessarily encompassing the entire student sample at our disposal, but distinct enough to indicate varying learning approaches. This distinction would allow us to ascertain the potential benefits of diverse adaptations, computer-aided support system configurations, or instructional methodologies tailored to each group's specific needs, if such measures were to be implemented.

Similar to the authors in [27] who explored K-12 mathematics, we discovered that there consistently exists a subset of students from which an effective model of student learning can be constructed to represent the entire available population. While this finding may appear inconclusive (or rather recurrent), it holds significant importance. It implies the absence of stereotypes such as "good students" or "bad students". Likewise, the concept of "fast learners" and "slow learners" may not hold true. Instead, we observe students approaching learning in distinct manners, with these variations being independent of the conventional dimensions we typically employ to measure learning.

10.2. How should our findings be interpreted

Mining behavior of students in problem-solving activities helped us find individual differences in how student solved programming problems. Certain students demonstrated a preference for extended contemplation, incorporating concepts that immediately improved code accuracy upon implementation. In contrast, other students tended to engage in more tinkering steps, involving the addition, removal, or modification of existing program code concepts that did not yield an immediate enhancement in code correctness. Indeed, for some students exhibiting the latter problem-solving behavior might be a sign of poor performance, but we should have this in mind that there are also students who have same behavior but are doing very well regarding their performance. This is an important finding, particularly for MOOCs design, as it points out that we should not necessarily intervene when students exhibit more tinkering steps. This is the way some students solve problems, and they can do well with that.

We attempted various simplified behavior-based grouping methods, but they did not yield noticeable variations in cross-prediction accuracies. In essence, there consistently exists a specific subset of students that contributes to a model applicable to the entire student population.

As a result, the feasibility of grouping students for targeted adaptation appears to be limited in its utility. Students, whether classified by their demographics, overall performance, or approach to solving programming problems, should not be treated as stereotypes, but rather considered in the context of a larger or smaller topic of the material, or even on the finer-grained level of a problem-solving session.

10.3. Potential limitations

We used traditional programming course and programming MOOC course data that come from the University in Europe. The education set-up in the country under study may diverge from the global norms. As a result, it is reasonable to consider that the composition of our student sample might have determined how assortments were made on basis of behaviors and performances. To ensure the validity and potential reaffirmation of our findings, we intend to pursue future research endeavors involving a larger and more representative student dataset, enabling us to reevaluate our analysis. Also, we plan to reduce the program concepts that we used in the modeling to only crucial ones. This would reduce inaccuracies that were due to noise in data.

References

[1] Ayres, J., Flannick, J., Gehrke, J. and Yiu, T., 2002. Sequential PAttern Mining Using a Bitmap Representation. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, KDD '02, p.429–435. Available from: https://doi.org/10.1145/775047.775109.

[2] Baker, R.S.J.D. and Yacef, K., 2009. The State of Educational Data Mining in 2009: A Review and Future Visions. *Journal of Educational Data Mining*, 1(1), p.3–17. Available from: https://doi.org/10.5281/zenodo.3554657.

[3] Breslow, L., Pritchard, D.E., DeBoer, J., Stump, G.S., Ho, A.D. and Seaton, D.T., 2013. Studying Learning in the Worldwide Classroom Research into edX's First MOOC. *Research & Practice in Assessment*, 8, pp.13–25. Available from: https://www.rpajournal.com/dev/wp-content/uploads/2013/05/SF2.pdf.

[4] Champaign, J., Colvin, K.F., Liu, A., Fredericks, C., Seaton, D. and Pritchard, D.E., 2014. Correlating Skill and Improvement in 2 MOOCs with a Student's Time on Tasks. *Proceedings of the First ACM Conference on Learning @ Scale Conference*. New York, NY, USA: Association for Computing Machinery, L@S '14, p.11–20. Available from: https://doi.org/10.1145/2556325.2566250.

[5] Corbett, A.T. and Anderson, J.R., 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4), pp.253–278. Available from: https://doi.org/10.1007/BF01099821.

[6] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R. and Lin, C.J., 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(61), pp.1871–1874. Available from: http://jmlr.org/papers/v9/fan08a.html.

[7] Ferguson, R. and Buckingham Shum, S., 2012. Social Learning Analytics: Five Approaches. *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*. New

York, NY, USA: Association for Computing Machinery, LAK '12, p.23–33. Available from: https://doi.org/10.1145/2330601.2330616.

[8] Fournier-Viger, P., Lin, J.C., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z. and Lam, H.T., 2016. The SPMF Open-Source Data Mining Library Version 2. In: B. Berendt, B. Bringmann, É. Fromont, G.C. Garriga, P. Miettinen, N. Tatti and V. Tresp, eds. *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part III*. Springer, *Lecture Notes in Computer Science*, vol. 9853, pp.36–40. Available from: https://doi.org/10.1007/978-3-319-46131-1_8.

[9] Glassman, E.L., Scott, J., Singh, R., Guo, P.J. and Miller, R.C., 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Transactions on Computer-Human Interaction*, 22(2), p.7. Available from: https://doi.org/10.1145/2699751.

[10] Guerra, J., Sahebi, S., Lin, Y. and Brusilovsky, P., 2014. The Problem Solving Genome: Analyzing Sequential Patterns of Student Work with Parameterized Exercises. In: J.C. Stamper, Z.A. Pardos, M. Mavrikis and B.M. McLaren, eds. *Proceedings of the 7th International Conference on Educational Data Mining, EDM 2014, London, UK, July 4-7, 2014*. International Educational Data Mining Society (IEDMS), pp.153–160. Available from: https://www.researchgate.net/publication/262967165.

[11] Guo, P.J. and Reinecke, K., 2014. Demographic Differences in How Students Navigate through MOOCs. *Proceedings of the First ACM Conference on Learning @ Scale Conference*. New York, NY, USA: Association for Computing Machinery, L@S '14, p.21–30. Available from: https://doi.org/10.1145/2556325.2566247.

[12] Hood, N. and Littlejohn, A., 2016. MOOC Quality: the need for new measures. *Journal of Learning for Development*, 3(3). Available from: https://doi.org/10.56059/jl4d.v3i3.165.

[13] Hosseini, R. and Brusilovsky, P., 2013. JavaParser: A Fine-Grain Concept Indexing Tool for Java Problems. In: E. Walker and C. Looi, eds. *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013, Memphis, USA, July 9-13, 2013*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 1009. Available from: https://ceur-ws.org/Vol-1009/0907.pdf.

[14] Hosseini, R., Vihavainen, A. and Brusilovsky, P., 2014. Exploring Problem Solving Paths in a Java Programming Course. In: B. du Boulay and J. Good, eds. *Proceedings of the 25th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2014, Brighton, UK, June 25-27, 2014*. Psychology of Programming Interest Group, p.9. Available from: https://ppig.org/papers/2014-ppig-25th-hosseini/.

[15] Huang, J., Piech, C., Nguyen, A. and Guibas, L.J., 2013. Syntactic and Functional Variability of a Million Code Submissions in a Machine Learning MOOC. In: E. Walker and C. Looi, eds. *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013, Memphis, USA, July 9-13, 2013*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 1009. Available from: https://ceur-ws.org/Vol-1009/0105.pdf.

[16] Murphy, C., Kaiser, G.E., Loveland, K. and Hasan, S., 2009. Retina: helping students and instructors based on observed programming activities. In: S. Fitzgerald, M. Guzdial, G. Lewandowski and S.A. Wolfman, eds. *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009, Chattanooga, TN, USA, March 4-7, 2009*. ACM, pp.178–182. Available from: https://doi.org/10.1145/1508865.1508929.

[17] Paas, F.G.W.C. and Van Merriënboer, J.J.G., 1993. The Efficiency of Instructional Conditions:

An Approach to Combine Mental Effort and Performance Measures. *Human Factors*, 35(4), pp.737–743. Available from: https://doi.org/10.1177/001872089303500412.

[18] Pavlik, P.I., Cen, H. and Koedinger, K.R., 2009. Performance Factors Analysis - A New Alternative to Knowledge Tracing. In: V. Dimitrova, R. Mizoguchi, B. du Boulay and A.C. Graesser, eds. *Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling, Proceedings of the 14th International Conference on Artificial Intelligence in Education, AIED 2009, July 6-10, 2009, Brighton, UK*. IOS Press, *Frontiers in Artificial Intelligence and Applications*, vol. 200, pp.531–538. Available from: https://doi.org/10.3233/978-1-60750-028-5-531.

[19] Perkins, D.N., Hancock, C., Hobbs, R., Martin, F. and Simmons, R., 1986. Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research*, 2(1), pp.37–55. Available from: https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL.

[20] Piech, C., Sahami, M., Koller, D., Cooper, S. and Blikstein, P., 2012. Modeling How Students Learn to Program. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, SIGCSE '12, p.153–160. Available from: https://doi.org/10.1145/2157136.2157182.

[21] Rivers, K. and Koedinger, K.R., 2013. Automatic Generation of Programming Feedback: A Data-Driven Approach. In: E. Walker and C. Looi, eds. *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013, Memphis, USA, July 9-13, 2013*. CEUR-WS.org, *CEUR Workshop Proceedings*, vol. 1009. Available from: https://ceur-ws.org/Vol-1009/0906.pdf.

[22] Sharma, K., Jermann, P. and Dillenbourg, P., 2015. Identifying Styles and Paths toward Success in MOOCs. In: O.C. Santos, J. Boticario, C. Romero, M. Pechenizkiy, A. Merceron, P. Mitros, J.M. Luna, M.C. Mihaescu, P. Moreno, A. Hershkovitz, S. Ventura and M.C. Desmarais, eds. *Proceedings of the 8th International Conference on Educational Data Mining, EDM 2015, Madrid, Spain, June 26-29, 2015*. International Educational Data Mining Society (IEDMS), pp.408–411. Available from: https://eric.ed.gov/?id=ED560766.

[23] Tóth, K., Rölke, H., Greiff, S. and Wüstenberg, S., 2014. Discovering Students' Complex Problem Solving Strategies in Educational Assessment. In: J.C. Stamper, Z.A. Pardos, M. Mavrikis and B.M. McLaren, eds. *Proceedings of the 7th International Conference on Educational Data Mining, EDM 2014, London, UK, July 4-7, 2014*. International Educational Data Mining Society (IEDMS), pp.225–228. Available from: https://www.dropbox.com/s/crr6y6fx31f36e0/EDM%202014%20Full%20Proceedings.pdf.

[24] Vihavainen, A., Vikberg, T., Luukkainen, M. and Pärtel, M., 2013. Scaffolding Students' Learning Using Test My Code. *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: Association for Computing Machinery, ITiCSE '13, p.117–122. Available from: https://doi.org/10.1145/2462476.2462501.

[25] Wang, X., Yang, D., Wen, M., Koedinger, K.R. and Rosé, C.P., 2015. Investigating How Student's Cognitive Behavior in MOOC Discussion Forum Affect Learning Gains. In: O.C. Santos, J. Boticario, C. Romero, M. Pechenizkiy, A. Merceron, P. Mitros, J.M. Luna, M.C. Mihaescu, P. Moreno, A. Hershkovitz, S. Ventura and M.C. Desmarais, eds. *Proceedings of the 8th International Conference on Educational Data Mining, EDM 2015, Madrid, Spain, June 26-29, 2015*. International Educational Data Mining Society (IEDMS), pp.226–233. Available from: https://eric.ed.gov/?id=ED560568.

[26] Wen, M. and Rose, C.P., 2014. Identifying Latent Study Habits by Mining Learner Behavior Patterns in Massive Open Online Courses. *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, CIKM '14, p.1983–1986. Available from: https://doi.org/10.1145/2661829.2662033.

[27] Yudelson, M., Fancsali, S., Ritter, S., Berman, S.R., Nixon, T. and Joshi, A., 2014. Better Data Beats Big Data. In: J.C. Stamper, Z.A. Pardos, M. Mavrikis and B.M. McLaren, eds. *Proceedings of the 7th International Conference on Educational Data Mining, EDM 2014, London, UK, July 4-7, 2014*. International Educational Data Mining Society (IEDMS), pp.205–208. Available from: https://www.researchgate.net/publication/279530590.

[28] Yudelson, M., Hosseini, R., Vihavainen, A. and Brusilovsky, P., 2014. Investigating Automated Student Modeling in a Java MOOC. In: J.C. Stamper, Z.A. Pardos, M. Mavrikis and B.M. McLaren, eds. *Proceedings of the 7th International Conference on Educational Data Mining, EDM 2014, London, UK, July 4-7, 2014*. International Educational Data Mining Society (IEDMS), pp.261–264. Available from: http://d-scholarship.pitt.edu/21833/.