A recommender system based on effort: towards minimising negative affects and maximising achievement in CS1 learning

Filipe D. Pereira[1], Hermino B. F. Junior[1], Luiz Rodriguez[4], Armando Toda[4], Elaine H. T. Oliveira[2], Alexandra I. Cristea[3], David B. F. Oliveira[2], Leandro S. G. Carvalho[2], Samuel C. Fonseca[2], Ahmed Alamri[3], and Seiji Isotani[4]

[1] Department of Computer Science, Federal University of Roraima, Boa Vista, Brazil
filipe.dwan@ufrr.br
[2] Institute of Computing, Federal University of Amazonas, Manaus, Brazil
[3] Department of Computer Science, Durham University, Durham, United Kingdom
[4] ICMC, University of Sao Paulo, Sao Carlos, Brazil

Abstract. Programming online judges (POJs) are autograders that have been increasingly used in introductory programming courses (also known as CS1) since these systems provide instantaneous and accurate feedback for learners' codes solutions and reduce instructors' workload in evaluating the assignments. Nonetheless, learners typically struggle to find problems in POJs that are adequate for their programming skills. A potential reason is that POJs present problems with varied categories and difficulty levels, which may cause a cognitive overload, due to the large amount of information (and choice) presented to the student. Thus, students can often feel less capable, which may result in undesirable affective states, such as frustration and demotivation, decreasing their performance and potentially leading to increasing dropout rates. Recently, new research emerged on systems to recommend problems in POJs; however, the data collection for these approaches was not fine-grained; importantly, they did not take into consideration the students' previous effort and achievement. Thus, this study proposes for the first time a prescriptive analytics solution for students' programming behaviour by constructing and evaluating an automatic recommender module based on students' effort, to personalise the problems presented to the learner in POJs. The aim is to improve the learners achievement, whilst minimising negative affective states in CS1 courses. Results in a within-subject double-blind controlled experiment showed that our method significantly improved positive affective states, whilst minimising the negatives ones. Moreover, our recommender significantly increased students' achievement (correct solutions) and reduced dropout and failure in problem-solving.

Keywords: online judge, data-driven analysis, recommender system

1 Introduction

Programming Online Judges (POJs) are automatic code correction environments that are typically used by students to improve their programming skills and/or train for programming competitions [41, 46, 26, 40, 47]. The adoption of these environments by instructors and institutions has increased in the last few years in introductory computing (so-called 'CS1') classes [41, 36]. Typically, in

educational scenarios, students code in an integrated development environment (IDE) tied to a POJ [7, 36]. Students then design their algorithms in the IDE and submit them to be evaluated by the POJ system, which provides them real-time feedback based on a case test analysis [41, 44, 7, 39, 34].

Alongside the growing popularity of POJs, data within theses systems are gaining attention [13, 6, 41, 46, 33, 14, 7, 28, 29, 39, 32, 31]. Despite the notorious benefits of POJs in education, these systems are not able to recommend the appropriate problems for the students, which may impact on affective perception, leading, over time, to affective states such as frustration [38, 45, 44, 8, 24]. Frustration has been shown to be directly related to the amount of effort a student needs to spend to solve a problem and may even lead to dropout [38, 25, 22, 1, 30]. This happens due to effort being intrinsically related to the students' confidence, competence and consequently affecting their motivation [19]. According to [5], the learners' effort can be measured by the amount of energy and time they expend to meet the academic requirements. [19] explain it is necessary to measure effort to assess students' motivation and satisfaction. Moreover, a good balance of effort required to solve tasks is related to increased achievement [12].

To adapt the programming problems to the students' effort, Recommender Systems (RS) appear as a viable solution [39]. RS are environments used to identify and provide content based on rules designed from user data. These systems have been widely used in educational scenarios [44, 2, 39]; however, few studies have tackled ways to provide recommendations based on a deep analysis of user behaviours. Specifically in the scope of programming learning, there are only a few studies available in the literature proposing methods to recommend problems in POJs, and such studies typically make the recommendations only based on students' attempts and results from the submissions to the POJ [44, 39]. Notice that a deep behavioural analysis of fine grained data is crucial to make appropriate recommendations [20].

As such, in this work, besides the variables previously used in the literature (attempts and results from submissions), we also track how students solve problems in the embedded IDE of a POJ and construct a holistic set of fine-grained features to represent the effort expected to solve a given problem. Using these features, we make a recommendation based on the following hypothesis: if a student s solves a given problem p (which we call a target problem), our method recommends a problem p' that requires an effort to be solved similar to that of p, assuming the student s would be able to solve the problem p'. Through exploring this hypothesis, we believe that the recommendations will minimise students' negative affective states, whilst maximising the positive ones, as the problems recommended will not require a disproportionate effort from the learners. In addition, as aforementioned, effort has been related to students' achievement in other fields beyond POJ [12, 34]. Hence, our second hypothesis is that our recommendation based on expected effort will increase the student achievement and decrease dropout and failure rate in problem-solving. Thus, this work aims at solving the following research question: *Does personalised recommendation based on effort influence the students' affect and achievement in online judges?*

2 Related Work

Programming is learned by doing, that is, students need to solve many problems to improve their skills and a POJ is a suitable tool for practising [41, 34].

However, given the huge amount of problems available in this system, frustration, confusion and other negative affective states might be triggered when learners are searching for problems or solving inadequate questions [39, 44]. Thus, in this section, we analyse studies that propose methods for the automatic recommendation of problems or pedagogical material in automatic assessment systems.

In this sense, [9] conducted a study mapping code submitted by a group of students to create a code profile that provided personalised instructions, which was based on previous recommendations made by humans. Through these instructions, the authors suggested that it was possible to reduce difficulties faced by students. They used a multi-label k-nearest neighbour technique to recommend a topic of programming for the student. For instance, after solving a given problem using the "if-then-else" structure, the recommender could suggest problems using "loops". However, this recommender presents a generic list of problems based on the topic. In this sense, students were still responsible for finding problems that they thought were closest to their programming skills. Here, using our hypothesis, our novel behavioural-based model recommends directly the problem to the student, not the programming topic. Moreover, POJ questions are typically not annotated with the topic and, hence, a human endeavour would be needed to apply manual annotations of topics [15].

Following, [17, 8] proposed to extract information from codes to choose the suggestions for students learning programming. [17] created an RS that provided hints during the solving process, using techniques such as *term frequency–inverse document frequency* to represent similarities. [8] created an RS that suggested learning materials to help teachers in designing courses. Different from [17] and [8] who focused on tips about learning materials, we used data-driven behaviour analysis to infer the knowledge and effort of the students based on the data logs generated through real-time execution.

[44] and [39] used learner behavioural data for automatic recommendation of problems in POJs. [44] proposed an RS that recommends problems based on a collaborative approach. They used a binary matrix as a basis for the recommender. [39] proposed a learning path recommendation system based on learner's submission history in an online judge. However, these authors [44, 39] considered only the number of attempts and results from the submissions as features. In this work, we extend the features that were used in previous works and others that were extracted from the codes submitted and fine-grained log data, including self-devised features. Moreover, different from all of them, we evaluated our method with real learners and checked their affective states and achievement rates when solving the problems.

In brief, none of the previous studies performed an analysis of effort considering such fine-grained set of features to design a behavioural recommender model, as a way to personalise the recommendations in POJs. Moreover, for the first time, to the best of our knowledge, our RS provides problems adapted to the students' skills, and we measured the resulting achievement rate and affective states when solving our recommended problems.

3 Materials and Methods

In this section, we present the methods and tools used in this study, describe the data collection process, feature extraction and architecture of the RS, as well

as the evaluation method for this study. Following, we present the instrument and data collection process:

- **Instrument:** We used a POJ called CodeBench[5], a home-made POJ system designed by the Institute of Computing team from Federal University of Amazonas, Brazil. Codebench allows teachers, instructors and lecturers to provide assignments for students to develop their programming skills. Moreover, students can perform self-direct learning. Once an answer for the problem is submitted, the system provides a real-time feedback. This system also includes an online IDE, where students can write and execute their codes. Some other features are management of classes, social interactions between students and lecturers, and learning materials sharing. All these characteristics are common in other POJs such as URI online judge [4], UVA online judge [37], and others.
- **Data collection**: We collected data from CS1 courses that were offered during 6 semesters (from 2016 to 2018). Students had to solve seven sets of exercises using Python. Each set of exercises is related to one of the following topics: (1) variables; (2) conditionals; (3) nested conditionals; (4) while loops; (5) vectors; (6) for loops; and (7) matrices. Furthermore, learners could solve other problems as wanted (self-direct learning). All the students solved the problems using the IDE provided in the POJ CodeBench. We collected data from 2,058 students that generated 535,619 code submissions to solve these exercises. Students' keystrokes were recorded in their logs on the server side. These log files are the sources of our extracted fine-grained features related to the students' behaviour to be used as input into our data-driven approach. With the data from the student's logs, we will represent the expected effort for the student to solve the programming problems.

3.1 Behaviour-based Recommender System based on effort

Students' effort is the amount of energy and time learners expend to meet the academic requirements [5]. In this work, we represent the student effort expected to solve a given problem as the aggregation of students' procedural and intellectual effort [5] combined with consolidated code metrics [27]. The procedural effort is related to how students succeeded (e.g. proportion of solved questions) in the assignment and the intellectual effort is related to how much energy the students used to solve the problems (e.g. number of attempts, time spent). The code metrics (e.g. number of loops) come from the software engineering field and are used to measure the learners' effort in building their code solutions.

Notice that effort is a psychological construct and, therefore, there is no standard way to measure it. In other words, there is no standardised scale that will measure how students efforts are used to solve problems [16]. In these cases, there is a need of features that may be used to indirectly measure one's effort. In addition, we need to define some observable, recordable measures that reflect the construct, called the *operational definition of the construct*. As such, we have established an operational definition to compute the expected effort to solve a given problem, using features that have already proved to be efficient in the literature [18, 42, 27, 44, 34, 7] and code metrics established in software engineering to measure the effort of programmers in the development process.

[5] codebench.icomp.ufam.edu.br

We extracted these features from the students' logs and codes and further processed them, e.g., extracting the average number of students' attempts for each problem, average number of lines of codes for each problem, etc. Thus, our observable, recordable measure to represent the effort is represented via the following features: *noAttempts* - proportion of users who didn't try to solve a given programming problem [43, 34]; *unsucNoRes* - proportion of users who failed to solve a given programming problem with a few attempts [43, 34]; *unsucRes* - proportion of users who tried hard, but failed to solve the problem [43, 34]; *sucNoRes* - proportion of users who solved the problem with a few attempts [43, 34]; *sucRes* - proportion of users who tried hard and managed to solve a given problem [34]; *attempts* - average of students' attempts to solve a given problem; *IDEUsage* - average resolution time of student on the Online IDE to solve a given problem [13, 34]; *contCicle* - average number of loops from submitted students' codes for a given problem [27]; *contCondition* - average number of conditional structures from submitted students' codes for a given problem [27]; *cyclomaticComplexity* - average cyclomatic Complexity from submitted students' codes for a given problem, where cyclomatic Complexity represent the source code as a control flow graph, corresponding to the number of independent paths of this graph [27]; *events* - average log lines of problems solved [34]; *nDistinctOperands* - average distinct arithmetic operands in the source codes; *nDistinctOperators* - average distinct arithmetic operators in the source codes; *quotientError* - average of repeated mistakes made by the students [18]; *quotientWatson* - attribution average penalties for mistakes made in a short period of time [42]; *sloc* - average lines of code sent in the online code edit [27]; *sucessAverage* - average problem submissions assessed as correct [13]; *test* - average number of times the student tested the source code [13]; *totalOperands* - average operands in the source codes; *totalOperators* - average operators present in the source codes; *variables* - average number of of variables in the source code [34].

We use such a large set of features to measure effort following [5], who explains that effort should be measured by a wide range of variables and expectations. Using this set of features forms the input data-driven behaviour for our Behaviour-based Recommender System (BRS) based on students' effort. The similarity between the recommended problem and the target problem is computed through nearest neighbour analysis, using cosine similarity as distance metric. We use this technique to support our first hypothesis that considers that a student s is able to solve a (recommended) problem p' of a same or similar level to a previously solved one p (target problem). As such, the nearest neighbour analysis is playing the role of matching the target and recommended problem by analysing the problems' similarities.

4 Evaluation of the Recommender Model

4.1 Participants

For the evaluation of our BRS, we recruited students who had already done introductory programming, from the Federal University of Roraima (UFRR), Brazil, due to convenience sampling, and since these students had already experience of learning with POJs, and could much easier and faster understand the purpose of the study. We have sent a message to computer science students from UFRR, explaining our research goals, asking for volunteers to participate in a 10-minute

phone call, scheduling calls for all who replied. Before the evaluation, we have explained the study to the learners and obtained their consent to participate. In total, 15 students agreed to participate of the experiment.

4.2 Measures

For each recommended problem, we asked the participants to make a comment about the effort required to solve the target problem and the recommended problem. Thus, we could evaluate manually the affective states within the comments based on the most frequent affective states when solving problems [10], which are boredom, confusion, engagement, neutral, and frustration. Besides that, in our context it is also crucial to evaluate when the learner is satisfied with the recommendation. Therefore, we included happiness, as [11] explain that happiness is a typical affective state presented when students are satisfied when solving problems. In Table 1, we summarise and describe all affective states used in our analysis.

Affective State	**Description**
Boredom	Uninterested in the current recommended problem.
Confusion	Poor comprehension of the problem, attempts to resolve erroneous belief.
Engagement	Student motivated to solve the current problem recommended.
Neutral	No visible affect, at a state of homeostasis.
Frustration	Problem recommended was not as expected, that is, more difficult or easier.
Happiness	Satisfaction with the recommendation, feelings of pleasure about the problem.

Table 1. Affective states used to evaluate learner's comments

We further use a data-driven approach to evaluate the recommendations in terms of achievement using the following metrics: i) achievement rate, which is the proportion of correct submissions over all submissions; ii) failure rate, which is the proportion of incorrect over all submissions; and iii) dropout rate, which is the proportion of recommended problems that were not attempted by the students over all problems. It is worth mentioning that students were free to execute and submit solutions for the recommended problems as many times as they wished, i.e., there was no limit of attempts set for recommended problems.

4.3 Experimental Manipulation

To evaluate the BRS itself (experimental treatment) we compared the personalised recommendation with a random recommender (control treatment). We called the second one Random Recommender System (RRS) because the input is known but not the output, which means that given a target problem, the RRS recommends the next problem(s) by performing a random selection of questions from pre-determined lists of problems selected by instructors. Instructors created these lists for students on CS1 courses, so they are real-life recommendations. The comparison between the BRS and RRS was conducted through a within-subject double-blind controlled experiment, where neither students nor authors knew which treatment (BRS or RRS) they were receiving.

Thus to conduct our experiment, we created two personalised lists of recommendations using each recommender method (BRS and RRS). Each personalised list comprises 8 problems, totalling 16 (2x8) problems per student, containing 12 recommendations and 4 target problems, totalling 180 recommendations (i.e., 12x15) to be evaluated. Each student had their own personalised list split into 2 groups, with each group containing 4 problems. The first group was composed of easy problems, whilst the second one of intermediate problems. The first question of each group was a target problem (TP_1 *and* TP_2) that was selected by the authors of this study in collaboration with lecturers and professors of programming. These target problems act as a starting point to balance the RS towards generating the recommendations, as a way to deal with the cold-start problem [21]. After the target problems, we have sequenced 3 automatic recommendations based on each target problem. Thus, we constructed the personalised list of recommended problems for the participants as follows: $TP_1 \rightarrow R_1, R_2, R_3$ and $TP_2 \rightarrow R_4, R_5, R_6$.

Doubtlessly, target problems were not included as part of the recommendation. For the easy target problems, we chose sequential and conditional problems (if...then...else), whereas for the intermediate problems we selected problems that use repetition structures (loops), vectors, strings and matrices. To personalise the recommendation for each student, we selected five different target problems for TP_1 and TP_2. Moreover, we calculated the 10 nearest neighbours for each target problem, so that we had 10 different recommendations for each target problem. After that, we randomly assigned 3 out of the 10 nearest neighbours of a given target problem to compose the above recommendations.

5 Results and Discussion

We performed a qualitative analysis of the students' comments[6] over the recommended problems to identify their affective states. To perform that analysis, two authors independently classified each comment based on the affective states presented in Table 1. Subsequently, we performed a Kappa Cohen Test to check the agreement level and, as a result, we achieved 0.83, which is considered a high level of agreement [3]. For the cases of disagreement, another author acted as the third judge. Using this classification, Figure 1 (left) shows the affective states present in the comments about the recommendations for each method. Comparing the methods, we can see a clear difference in terms of happiness and frustration (as affective measures - see Section 4.2). Indeed, the difference is statistically significant ($p - value < 0.05$, χ_2 - even after Bonferroni correction), which reveals that our method maximises the positive affective state (happiness, related to satisfaction), whilst minimising frustration.

Analysing each affective state in isolation, we observe only few neutral comments, which makes sense, since the students' comments about the recommendations tend to be pragmatic, that is, they usually stated that they were satisfied with the recommendation (happiness) or that the recommendations did not require the same effort as the target problems (frustration). Moreover, we can observe that boredom and engagement were not assigned for any comment. A possible reason is that the students may not have experienced an aversive state

[6] www.dropbox.com/s/uxkvhohvuo1itmq/comments_english.xlsx?dl=0

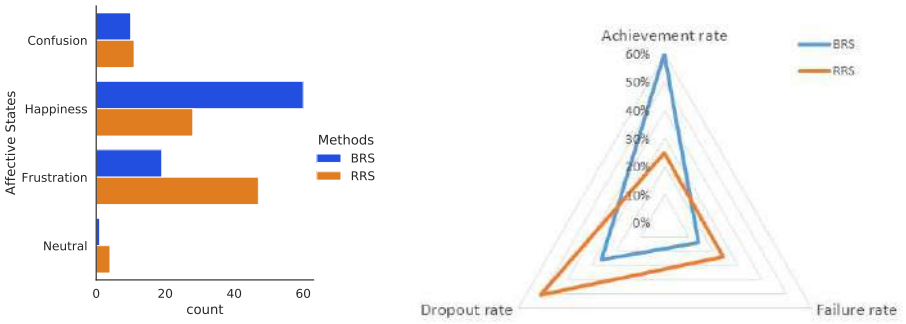

Fig. 1. Affective states classified based on learners' comments (left) and analysis of the recommendations submitted to the POJ (right).

to the activity nor have felt sufficiently engaged as they treated the recommendations as an experiment and not as an usual learning activity.

Another affective state that occurred with a relatively low frequency (N = 21) was confusion. In total, there were 11 cases of confusion in the RRS and 10 cases in the BRS, which reveals a balance in relation to this state. This affective state occurred when students did not understand the problem statement, or the way in which the outputs of their codes should be presented in order to pass, for each test case. To illustrate, in some comments, students reported that their codes were correct, but the POJ did not judge them right because, apparently, the test cases were pointing out an error that they could not find. [41] state that this phenomenon can happen, as test cases are analysed by comparing strings, so if the student forgot a line break in a *print* command, then the problem can be assessed as wrong, even with the logic being right. Nonetheless, notice that this is a limitation of the way in which POJs assess exercises and not a limitation of our recommender method. Similarly, a poorly designed question (which can cause confusion) is out of the scope of our method. Still, these cases of confusion may have slightly influenced the failure rate and dropout in both methods. However, as there are almost the same number of confusion cases for both methods, their influence potentially weighed equally.

For happiness, there are 28 cases in our baseline, whereas 60 in our method, more than double. In addition, there were 47 cases of frustration in the RRS, whilst 19 in our method. This is a first evidence that our method is mitigating frustration, whilst maximising the students' satisfaction (i.e. happiness), supporting our first hypothesis that the recommendations will minimise students' negative affective states, whilst maximising the positive ones, as the problems recommended will not require a disproportionate effort from the learners.

After this qualitative analysis of the student affective states, we analysed the achievement of learners when solving the recommended problems. Figure 1 (right) shows the results for each method in terms of three rates: *achievement, failure and dropout*. When the students solved the problems recommended by the BRS, 60% of their submissions were assessed as correct (achievement rate), whereas using the RRS, the achievement rate was of only 25%. In terms of failure rate, only 14% of students' solutions were not accepted within the BRS, against 24% within the RRS. Indeed, these differences are statistically significant ($p - value < 0.05$ - $\chi^2 test$, even after Bonferroni correction). Thus, these results indicate that for the RRS, the effort required to solve these problems is

much higher than that used for the target problems. Furthermore, this evidence suggests the importance of recommending problems that are more appropriate to the students' efforts, so as not to lead them to a low achievement rate and, hence, to frustration. Additionally, something worth noting is that for the RRS, students had a high rate of untried problems (51% of dropout rate), whereas for the BRS this was only 26%. These findings support our second hypothesis that recommendation based on effort expected will increase the student achievement and decrease dropout and failure rate.

About the difference in terms of dropout and failure rate, we can state that this is another confirmation that the problems recommended by the RRS either required more effort from learners or were more complex to the point where the students did not even try to solve them. Such high dropout rate from the RRS is a clear evidence of students' frustration in trying to solve problems not adequate to the effort expected, here, the one applied to the target problem. Other reasons that may have led the student not to try may be either the lack of understanding of the problem (confusion) or the lack of skills. Nonetheless, the target problems for each method are, respectively, an easy and intermediate problem. As such, if the RS works well, the first group of recommendations should comprise only easy problems, and the second group of recommendations should contain only intermediate problems. Consequently, the lack of skills to solve the problem should not have been present, as all the students who solved the recommended problems (via both methods) had already done introductory programming and were able to solve easy and intermediate problems. So, what likely happened was some bad recommendations in both systems, proportional to the students' dropout rate and failure rate. Notice that BRS was statistically superior in terms of achievement rate, failure rate and dropout rate, which likely means that the recommendation of the BRS were more suitable to students effort.

6 Pedagogical Implications

In summary, it is worth noting that our BRS shows potential to support programming classes for learners and for instructors who use POJs.

For the student, our recommender mitigates the burden of searching for problems that are adequate to their knowledge level and skills, capable of enabling self-directed learning in POJs. Moreover, our results showed that students felt less frustration and more happiness when completing assignments recommended by the BRS. Mitigating and improving frustration and happiness, respectively, is important to improve learning outcomes. Thus, this finding implies the usefulness of our proposed recommendation approach shows its potential to enhance learning experiences in solving programming assignments. Additionally, our finding about the students' achievement implies the stringent need to provide adequate recommendations for programming students to practice.

Finally, instructors typically need to create variations of programming assignments lists for different classes, in order to avoid plagiarism, for example. Using our method, considering each problem in a list of exercises already created by a instructor as a target problem. By generating N recommendations for each of these problems, we can automatically compose N new lists of exercises that require effort and knowledge similar to those required to solve the original. Thus, *the instructor's workload to design new programming assignment lists is significantly reduced.*

7 Conclusions, Limitations and Future Works

The evidence we found in our qualitative analysis is aligned with the achievement rate analysis, supporting our hypotheses that, in general, new recommendations require a similar level of effort to the target problem. Indeed, the higher level of frustration in our baseline is potentially the driving factor that lead to such a high dropout and failure rate in problem-solving, whereas the higher rate of happiness might be related to the high achievement rate in our method. Thus, supporting our second hypothesis that the affective states influences achievement, defined here in terms of lower failure and dropout and higher number of problems solved.

Notice that human responses may be subject to bias [5], as it is difficult to control human attitudes and behaviour, even in a controlled experiment. Thus, the way we evaluated our recommendation method was designed to reduce potential biases. That is, besides the comments analysis, we also evaluated the students' interaction with the POJ and the problem solving process. In future works we envision to evaluate the effect of our methods for teaching and learning introductory programming by employing our method in real CS1 classes.

Finally, effort required to solve a given problem depends on the previous knowledge acquired by the learner about the topic of that problem. To illustrate, if the student already knew how to manipulate vectors using Python, it is easier for them to code a vector sum, as the *numpy* module allows summing up vectors as scalars. However, for a student who had no prior knowledge of vector manipulation with *numpy*, the effort to learn would be greater. The way effort was modelled in our BRS does not take into account this prior knowledge that the student would have about the topic. Thus, a potential limitation of our BRS is recommending a vector sum problem for a student who is learning how to sum scalars. As a way to solve that problem, in future works we envision to take into consideration topics of problems, and merge this study with other work we have on the topic of automatic detection [35]. Additionally, according to our first hypothesis, the problems recommended by our BRS tend to require similar effort of the target problem. However, students would not progress if the effort required to solve the next recommended problems does not increase. To deal with this, we also intend to merge this study with our work about detecting the difficult level of programming problems [23], so that we can create a mechanism to progressively increase the effort required to solve problems when making recommendations.

Acknowledgements

This research, carried out within the scope of the Samsung-UFAM Project for Education and Research (SUPER), according to Article 48 of Decree nº 6.008/2006 (SUFRAMA), was partially funded by Samsung Electronics of Amazonia Ltda., under the terms of Federal Law nº 8.387/1991, through agreements 001/2020 and 003/2019, signed with Federal University of Amazonas and FAEPI, Brazil. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq grant 308513/2020-7).

References

1. Alamri, A., Alshehri, M., Cristea, A., Pereira, F.D., Oliveira, E., Shi, L., Stewart, C.: Predicting moocs dropout using only two easily obtainable features from the first week's activities. In: International Conference on Intelligent Tutoring Systems. pp. 163–173. Springer (2019)
2. Aljohani, T., Pereira, F.D., Cristea, A.I., Oliveira, E.: Prediction of users' professional profile in moocs only by utilising learners' written texts. In: International Conference on Intelligent Tutoring Systems. pp. 163–173. Springer (2020)
3. Artstein, R., Poesio, M.: Inter-coder agreement for computational linguistics. Educational and Psychological Measurement 20(1):37-46 (2008)
4. Bez, J.L., Tonin, N.A., Rodegheri, P.R.: Uri online judge academic: A tool for algorithms and programming classes. In: 2014 9th International Conference on Computer Science & Education. pp. 149–152. IEEE (2014)
5. Carbonaro, W.: Tracking, students' effort, and academic achievement. Sociology of Education **78**(1), 27–49 (2005)
6. Caro-Martinez, M., Jimenez-Diaz, G.: Similar users or similar items? comparing similarity-based approaches for recommender systems in online judges. In: International Conference on Case-Based Reasoning. pp. 92–107. Springer (2017)
7. Carter, A., Hundhausen, C., Olivares, D.: Leveraging the ide for learning analytics. The Cambridge Handbook of Computing Education Research, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge pp. 679–706 (2019)
8. Chau, H., Barria-Pineda, J., Brusilovsky, P.: Content wizard: concept-based recommender system for instructors of programming courses. In: Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization. pp. 135–140 (2017)
9. De Oliveira, M.G., Ciarelli, P.M., Oliveira, E.: Recommendation of programming activities by multi-label classification for a formative assessment of students. Expert Systems with Applications **40**(16), 6641–6651 (2013)
10. D'Mello, S., Calvo, R.A.: Beyond the basic emotions: what should affective computing compute? In: CHI'13 extended abstracts on human factors in computing systems, pp. 2287–2294 (2013)
11. D'Mello, S.K., Lehman, B., Person, N.: Monitoring affect states during effortful problem solving activities. International Journal of Artificial Intelligence in Education **20**(4), 361–389 (2010)
12. Duckworth, A.L., Eichstaedt, J.C., Ungar, L.H.: The mechanics of human achievement. Social and Personality Psychology Compass **9**(7), 359–369 (2015)
13. Dwan, F., Oliveira, E., Fernandes, D.: Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). vol. 28, p. 1507 (2017)
14. Fonseca, S., Oliveira, E., Pereira, F., Fernandes, D., de Carvalho, L.S.G.: Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). vol. 30, p. 1651 (2019)
15. Fonseca, S.C., Pereira, F.D., Oliveira, E.H., Oliveira, D.B., Carvalho, L.S., Cristea, A.I.: Automatic subject-based contextualisation of programming assignment lists. EDM (2020)
16. Haden, P.: Descriptive statistics. The Cambridge Handbook of Computing Education Research, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge pp. 102–131 (2019)
17. Hosseini, R., Brusilovsky, P.: A study of concept-based similarity approaches for recommending program examples. New Review of Hypermedia and Multimedia **23**(3), 161–188 (2017)

18. Jadud, M.C.: Methods and tools for exploring novice compilation behaviour. In: Proceedings of the second international workshop on Computing education research. pp. 73–84. ACM (2006)
19. Keller, J.M.: Motivational design for learning and performance: The ARCS model approach. Springer Science & Business Media (2009)
20. Kulkarni, P.V., Rai, S., Kale, R.: Recommender system in elearning: A survey. In: Proceeding of International Conference on Computational Science and Applications. pp. 119–126. Springer (2020)
21. Lam, X.N., Vu, T., Le, T.D., Duong, A.D.: Addressing cold-start problem in recommendation systems. In: Proceedings of the 2nd international conference on Ubiquitous information management and communication. pp. 208–211 (2008)
22. Lee, D.M.C., Rodrigo, M.M.T., d Baker, R.S., Sugay, J.O., Coronel, A.: Exploring the relationship between novice programmer confusion and achievement. In: International conference on affective computing and intelligent interaction. pp. 175–184. Springer (2011)
23. Lima, M., de Carvalho, L.S.G., de Oliveira, E.H.T., Oliveira, D.B.F., Pereira, F.D.: Classificação de dificuldade de questões de programação com base em métricas de código. In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. pp. 1323–1332. SBC (2020)
24. Luxton-Reilly, A., Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., Szabo, C.: Introductory programming: a systematic literature review. In: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. pp. 55–106 (2018)
25. Ngai, G., Lau, W.W., Chan, S.C., Leong, H.v.: On the implementation of self-assessment in an introductory programming course. ACM SIGCSE Bulletin **41**(4), 85–89 (2010)
26. de Oliveira, J., Salem, F., de Oliveira, E.H.T., Oliveira, D.B.F., de Carvalho, L.S.G., Pereira, F.D.: Os estudantes leem as mensagens de feedback estendido exibidas em juízes online? In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. pp. 1723–1732. SBC (2020)
27. Otero, J., Junco, L., Suarez, R., Palacios, A., Couso, I., Sanchez, L.: Finding informative code metrics under uncertainty for predicting the pass rate of online courses. Information Sciences **373**, 42–56 (2016)
28. Pereira, F.D., Oliveira, E.H., Fernandes, D., Cristea, A.: Early performance prediction for cs1 course students using a combination of machine learning and an evolutionary algorithm. In: 2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT). vol. 2161, pp. 183–184. IEEE (2019)
29. Pereira, F., Oliveira, E., Fernandes, D., de Carvalho, L.S.G.C., Junior, H.: Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). vol. 30, p. 1451 (2019)
30. Pereira, F.D., Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A., Alshehri, M.: Early dropout prediction for programming courses supported by online judges. In: International Conference on Artificial Intelligence in Education. pp. 67–72. Springer (2019)
31. Pereira, F.D., Toda, A., Oliveira, E.H., Cristea, A.I., Isotani, S., Laranjeira, D., Almeida, A., Mendonça, J.: Can we use gamification to predict students' performance? a case study supported by an online judge. In: International Conference on Intelligent Tutoring Systems. pp. 259–269. Springer (2020)
32. Pereira, F.D., Fonseca, S.C., Oliveira, E.H., Oliveira, D.B., Cristea, A.I., Carvalho, L.S.: Deep learning for early performance prediction of introductory programming

students: a comparative and explanatory study. Brazilian journal of computers in education. **28**, 723–749 (2020)
33. Pereira, F.D., Oliveira, E.H.T., Oliveira, D.F.B.: Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código. Mestrado em informática, Universidade Federal do Amazonas, Manaus (2018)
34. Pereira, F.D., Oliveira, E.H., Oliveira, D., Cristea, A.I., Carvalho, L., Fonseca, S., Toda, A., Isotani, S.: Using learning analytics in the amazonas: understanding students' behaviour in cs1. British journal of educational technology. (2020)
35. Pereira, F.D., Pires, F., Fonseca, S.C., Oliveira, E.H.T., Carvalho, L.S.G., Oliveira, D.B.F., Cristea, A.I.: Towards a human-ai hybrid system for categorising programming problems. SIGCSE '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3408877.3432422, https://doi.org/10.1145/3408877.3432422
36. Pereira, F.D., de Souza, L.M., de Oliveira, E.H.T., de Oliveira, D.B.F., de Carvalho, L.S.G.: Predição de desempenho em ambientes computacionais para turmas de programação: um mapeamento sistemático da literatura. In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. pp. 1673–1682. SBC (2020)
37. Revilla, M.A., Manzoor, S., Liu, R.: Competitive learning in informatics: The uva online judge experience. Olympiads in Informatics **2**(10), 131–148 (2008)
38. Rodrigo, M.M.T., Baker, R.S.: Coarse-grained detection of student frustration in an introductory programming course. In: Proceedings of the Fifth International Workshop on Computing Education Research Workshop. p. 75–80. ICER '09, Association for Computing Machinery, New York, NY, USA (2009)
39. Saito, T., Watanobe, Y.: Learning path recommendation system for programming education based on neural networks. International Journal of Distance Education Technologies (IJDET) **18**(1), 36–64 (2020)
40. dos Santos, I.L., Oliveira, D.B.F., de Carvalho, L.S.G., Pereira, F.D., de Oliveira, E.H.T.: Tempos de transição em estados de corretude e erro como indicadores de desempenho em juízes online. In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. pp. 1283–1292. SBC (2020)
41. Wasik, S., Antczak, M., Badura, J., Laskowski, A., Sternal, T.: A survey on online judge systems and their applications. ACM Computing Surveys (CSUR) **51**(1), 1–34 (2018)
42. Watson, C., Li, F.W., Godwin, J.L.: Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In: 2013 IEEE 13th international conference on advanced learning technologies. pp. 319–323. IEEE (2013)
43. Yera, R., Martínez, L.: A recommendation approach for programming online judges supported by data preprocessing techniques. Applied Intelligence **47**(2), 277–290 (2017)
44. Yera Toledo, R., Caballero Mota, Y., Martínez, L.: A recommender system for programming online judges using fuzzy information modeling. In: Informatics. vol. 5, p. 17. Multidisciplinary Digital Publishing Institute (2018)
45. Yu, R., Cai, Z., Du, X., He, M., Wang, Z., Yang, B., Chang, P.: The research of the recommendation algorithm in online learning. International Journal of Multimedia and Ubiquitous Engineering **10**(4), 71–80 (2015)
46. Zhao, W.X., Zhang, W., He, Y., Xie, X., Wen, J.R.: Automatically learning topics and difficulty levels of problems in online judge systems. ACM Transactions on Information Systems (TOIS) **36**(3), 27 (2018)
47. Zordan Filho, D.L., de Oliveira, E.H.T., de Carvalho, L.S.G., Pessoa, M., Pereira, F.D., de Oliveira, D.B.F.: Uma análise orientada a dados para avaliar o impacto da gamificação de um juiz on-line no desempenho de estudantes. In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. pp. 491–500. SBC (2020)