I know what you coded last summer

Lucas Mendonça de Souza[1], **Igor Moreira Félix**[1], **Bernardo Martins Ferreira**[1], **Anarosa Alves Franco Brandão**[2], **Leônidas de Oliveira Brandão**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

[2]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

lucasmens@ime.usp.br, leo@ime.usp.br,
igormf@ime.usp.br, anarosa.brandao@poli.usp.br

Abstract. *The outbreak of the COVID-19 pandemic caused a surge in enrollments in online courses. Consequently, this boost in number of students affected teachers' ability to evaluate exercises and resolve doubts. In this context, tools designed to evaluate and provide feedback on code solutions can be used in programming courses to reduce teachers workload. Nonetheless, even when using such tools, the literature shows that learning how to program is a challenging task. Programming is complex and the programming language employed can also affect students outcomes. Thus, designing good exercises can reduce students difficulties in identifying the problem and help reduce syntax challenges. This research employs learning analytics processes on automatic evaluation tools interaction logs and code solutions to find metrics capable of identifying problematic exercises and their difficulty. In this context, an exercise is considered problematic if students have problems interpreting its description or its solution requires complex programming structures like loops, conditionals and recursion. The data comes from online introductory programming courses. Results show that the computed metrics can identify problematic exercises, as well as those that are being challenging.*

1. Introduction

During COVID-19 pandemic online courses have seen a surge in enrollments, especially in its early months, from March to June, when most were in lockdown at home [Impey and Formanek 2021]. According to Impey and Formanek, this surge was most significant with learners aged 18 to 30 looking to get ahead in their careers. This growth impacts teachers' ability to evaluate large quantities of exercises. The use of automatic assessment tools can decrease such impact. For instance, in programming courses, such tools not only help teachers, but also give to students fast and consistent feedback without the need of waiting for someone to look at their code and point out errors [Ala-Mutka 2005, Pears et al. 2007].

Easing the learning of how to program becomes ever more important with computing spread in society. In fact, Papert discussed the importance of teaching kids how to program, so they would not be programmed [Papert 1980]. Since then, studies reported how difficult learning programming can be. Not only introductory programming courses have high failure rates, but also the complexity of languages syntax hampers the learning process [Bosse and Gerosa 2015, Caspersen 2007]. Another difficulty students face is identifying the problem and the steps needed to solve it [Sleeman 1986, Gomes and Mendes 2007]. Considering these difficulties, exercises can be built in a way to provide students with a stepwise learning experience [Wang and Wong 2008]. So, based on three different introductory programming courses, this work aims to find metrics that can help teachers to identify problematic exercises and how difficult they are.

2. Background

In this section we discuss the underlying theory behind this research.

2.1. Introductory Programming Difficulties

Du Boulay stated in 1986 that "Learning to program is not easy". Although that statement is not recent, the numbers confirm that this difficulty is still alive. For instance, failure and dropout rates of an introductory programming course in the last three years of the Summer Program at Universidade de São Paulo provide some evidence for Du Boulay's statement. The rates achieved 31% in 2019, 58% in 2020 and 51% in 2021.

These failure rates are frequently reported by teachers and researchers. For instance, Bosse and Gerosa reported in 2015 that more than 50% had failed, during the years of 2010 to 2014 in an introductory programming course. These high failure rates expose how difficult is learning to program.According to [Caspersen 2007], understanding the programming process and how to transfer acquired skills are the main challenges faced by students. Other studies found that the programming language employed in the teaching process can also affect students results and can be a source of confusion for the learners [Gomes and Mendes 2007, Lahtinen et al. 2005, Caspersen 2007]. There are also problems related to misunderstanding of programming structures, i.e. loop and conditionals, that leads to students producing wrong solutions [Lahtinen et al. 2005, Milne and Rowe 2002]. Moreover, studying habits and teaching methodology were also found to impact their results [Gomes and Mendes 2007, Caspersen 2007].

2.2. Learning Analytics

The use of learning management systems (LMS) in the educational environment like a virtual classroom makes online teaching and learning possible. In this virtual space, teachers and students can access lots of resources, like didactic content, questionnaires, chats, activities and forums. All these resources generate large sets of data, representing the learning steps and users' interaction, that are captured and stored by LMS, usually in relational databases. Such data can be analyzed for different proposes, *e.g.* in the early identification of students in risk of failure [Akçapınar et al. 2019, Félix et al. 2017, Teodoro and Kappel 2020]; some pattern identification in the learning behavior to predict final exam grades [Blikstein et al. 2014]; and also providing support to group formation in Computer-Supported Collaborative Learning (CSCL) [Liang et al. 2021].

This exploration of educational data has been identified as learning analytics, defined as "measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" [Siemens and Gasevic 2012]. In this context, this research applied learning analytics to identify and classify (i) the exercises difficulty; (ii) the students behavior profile; and (iii) their correlation to students' grade. This approach allows not only an easy visualization of the data but also can support more sound predictions to be made [Chaturvedi 2017, Aleem and Gore 2020], improving learning outcomes.

2.3. Automatic Evaluation of Programs

An algorithm can be defined as a sequence of computational steps that transform an input into an output [Cormen et al. 2009]. So, it can be seen as a tool designed to solve a well-specified problem. Considering the specification, it's possible to use automatic evaluation tools to assess if some code solves correctly the problem. According to Ala-Mutka and Pears, automatic evaluation is a valuable tool for both students and lecturers [Ala-Mutka 2005, Pears et al. 2007]. They provide fast and consistent feedback for learners to identify their mistakes, while releasing teachers from grading [Ala-Mutka 2005].

Besides that, since they can evaluate large amounts of submissions with quick feedback, automatic evaluation tools can reduce lectures and instructors workload [Ihantola et al. 2010, Ala-Mutka 2005]. One of the most common methods of automatic evaluation is output matching [Ala-Mutka 2005]. This method treats the code as a black-box giving it a series of inputs and assessing only its outputs by comparing them with the ones specified by the teacher as expected outputs [Arifi et al. 2015]. The final result for each test-case, a pair of inputs and outputs, is binary, and the answer provided by the student's program is either "correct" or "incorrect" [Arifi et al. 2015].

On the one hand, an important advantage of output matching evaluation is that it doesn't allow any deviation from the expected output. In fact, the program answer for a given input must be exactly the same as the specified by the teacher while creating the exercise [Ihantola et al. 2010]. On the other hand, the output matching leads to a situation of a very unforgiving evaluation because, even though a human can easily ignore irrelevant mistakes, such as misspelled words, an automatic tool may not be able to do the same [Ala-Mutka 2005].

3. Related Work

Different techniques, methods and datasets have been used to analyze difficulties in introductory programming activities and exercises. [Lima et al. 2020] proposed a method to classify the difficulties in introductory programming exercises, according to solutions previously registered by teachers. The researchers found that it's possible to estimate the questions' difficulties using metrics extracted from code submitted by students. These metrics include, for instance, the amount of variables, attributions, logical and arithmetic operators and others. Besides that, according to authors, the difficulty level, is associated with the *success rate* of students' submissions. Our approach uses only students solutions as source of data. Moreover, we attempt to mitigate the influence of students long intervals between submissions by deriving a new metric based on code change and a formula to calculate the average grade (see Section 5.1.2).

In another work [Effenberger et al. 2019], an investigation of the classification of programming problems was based on statistics using the submissions' logs. Through clustering techniques (Fuzzy C-mean cluster), problems were categorized in difficulty levels i.e. easy, medium and hard. According to [Pelánek et al. 2021], such categorization must be distinguished in two different metrics i.e. the complexity and the difficulty. In our approach, although students submitted code is analyzed along side with interactions logs, we do not analyze code structure like presence of loop or flow control commands in the code. Moreover, this research considers the code as whole and how it changes over time as a potential metric for exercise difficulty and complexity.

4. Methodology

An exploratory analysis on students' submissions log, code solutions and grades, was conducted to identify potential metrics for classifying exercises.The code solutions are programming exercises that can be submitted at least in a week. All the exercises grade counts to the final course grade. No plagiarism check was made since the problems were straight forward, submissions varying from an average of 6.8 to 43.3 lines of code in length. This prevent us running plagiarism check with confidence. To support data exploration and analysis, four main phases were considered: (i) data collection; (ii) data preprocessing; (iii) analysis; and (iv) results interpretation.

4.1. Participants

Participants of this study were enrolled in two courses:one in the context of the Introductory Programming course of the 2021 Summer Program, and the other a distance learning CS1 course for undergrad students who failed the in-person version, lectured at IME-USP. Participants of the Summer Program simply want to learn about programming and how to program. The Summer Program had morning and night classes, running online only due to COVID-19. Moodle was the adopted LMS. CS1 had run in the 1st semester of 2019.

Participants were divided into three groups: G1 - 37 students of the morning classes; G2 - 67 students of the night classes; and G3 - 109 students of the CS1.

Most of the summer students have no prior experience in programming. Differently, all CS1 students have previous experience with programming and the course content.

4.2. Automatic Evaluation Tools

In the summer courses it was adopted two different programming tools integrated to Moodle to solve exercises proposed by teachers: iVProg (Interactive Visual Programming on the Internet) and VPL (Virtual Programming Lab). Both tools provide automatic evaluation and are free software. iVProg[1] is a free educational software, to support teaching and learning of introductory programming in which novices interact with graphic elements to build their algorithms [Felix et al. 2019].

VPL is a Moodle plugin used to automatic evaluate code written in different programming languages (e.g., C, Python and Java). One of its main components is an embedded code editor, in which students can program in any enabled programming language [Rodríguez del Pino et al. 2010]. Different from iVProg, in VPL it is possible to students to use other tools to write and tests their solutions before submitting them to the VPL plugin. Both iVProg and VPL can be used to create automatic evaluated exercises using output matching (see section 2.3). As each test-case yields a binary evaluation, the grade in an exercise is the ratio of positive evaluations and the total available cases.

[1]*iVProg*: available at `https://www.usp.br/line/ivprog/`

5. Dataset

The dataset is composed of logs from students interactions with the adopted programming tools. The interaction data log consists of a total of 122,432 entries from both tools, where 29,850 entries come from G1, 37,829 from G2 and 83,818 from G3.

Code solutions and grades assigned by each tool were also analyzed. A total of 39,183 code solutions were analyzed, being 7,158 from G1, 8,594 from G2 and 23,431 from G3, being G1 and G2 solutions coded in C and G3 solutions coded in Python and C.

5.1. Preprocessing & Data Transformation

In order to analyze the log data collected from the programming tools, some preprocessing was required to extract and normalize the information needed. First, map all grades of VPL to the interval $[0, 1]$. Second, VPL log timestamps were transformed from a string representation to a UNIX epoch number, because VPL counts minutes instead of seconds. As such, we spread similar timestamps evenly, using 30 or 20 seconds steps if more than two were similar. Moreover, all C and Python code went through a preprocessing to reduce the effects of submissions some students make to organize their code. Thus, all variables (including functions parameters) in the code were renamed to the form v_i, where $i \in N$ is the position of the variable in the text code. This preprocessing step also removed blank spaces and comments. This step was not required for iVProg code since it generates the final code, reducing the effects of students formatting styles.

5.1.1. First data transformation step

After all preprocessing steps, we then proceeded to extract information regarding each (i) time window between submissions (TES); (ii) code changing between submissions (DES); and (iii) the grade assigned by the automatic evaluator. We also used the data from (ii) and (i) to calculate the ratio $DT = DES/TES$. This transformation step and extracting process is described in the following paragraphs.

For each submission, TES was calculated by subtracting from the timestamp of a given submission, the timestamp of the submission before it. Since the first submission do not have a submission before it, we decided to use the student first interaction with the tool in each exercise as a starting point. However, each tool registered their events in a different manner. For iVProg, the starting point is the user first click, since it records every user interaction. Diversely, VPL only registers user clicks on specific links. As the logs were ordered in ascending order by the timestamp, we searched the logs for the event representing the student submitting a solution to the exercise for the first time. Then we looked backwards to find the closest event regarding the student reading the exercise description and used its timestamp as the starting point.

DES was tracked by using the Levenshtein distance of the code from a given submission to the code from the previous one. Preprocessing the code reduced an increasing distance between text codes that only changes formatting or variables naming. Regarding the first submission, the empty string was used as the previous submission code.

The submission's grade remained as they were in the logs. Computing DT was straight forward once we had TES and DES. This metric represents the number of changes per

Table 1. Metadata structure generated during first data transformation phase

Metadata	Description
student_id	Moodle student ID associated with the submission
TES	time window for the submission
DES	Levenshtein distance between the submission code and the previous one
DT	DES/TES, number of changes in the submission code per seconds
grade	Grade associated with the submission
timestamp	Original timestamp associated with the submission

second the student performed in this submission. Other data already present in the logs were also preserved: the student ID and the original timestamp of the submission. The final data structure contains all metadata presented in Table 1.

5.1.2. Second data transformation step

During the second transformation step, we used the data from the first step to generate new metadata about all students' submissions. For each student who submitted a given exercise, we generated: the highest time window between submissions ($MTES$); the highest number of code modifications ($MDES$); and the average submission grade (DEX).

In order to calculate $MTES$ for a given exercise, we simply compute the highest TES for each student in that exercise. Computing $MDES$ follows the same approach, but instead of TES, DES is used. The average grade DEX is calculated by using the following equation:

$$DEX = \frac{\bar{g}}{TMS + n} \tag{1}$$

where $\bar{g}$ is the student's average grade for that exercise, n is the number of submissions from that student and TMS is defined as:

$$TMS = \frac{last\ timestamp\ - first\ timestamp}{2} \tag{2}$$

Table 2 presents the metadata data structure generated after the second data transformation.

Table 2. Metadata structure generated during second data transformation phase

Metadata	Description
student_id	Moodle student ID associated with the submission
MTES	Highest TES
MDES	Highest DES
DEX	Average grade computed using equation (1)

5.2. Analysis

Given the exploratory nature of this research, the data analysis was approached as an iterative and cyclic process. Taking advantage of the data preprocessing and transformation for an easier visualization, we analyzed some of the results generated from the first transformation step to check the students submission behavior. We noticed that DT plot from students scores could have quite different shapes (Figures 1 and 2).

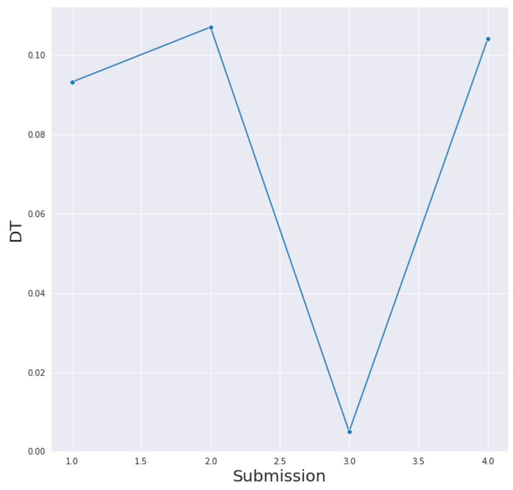

Figure 1. Student A DT plot for 4 submission until max grade

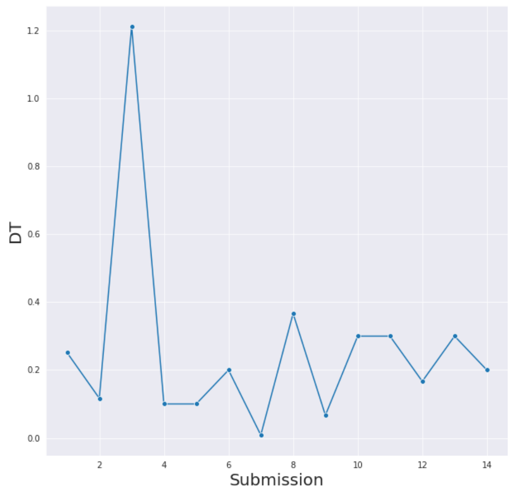

Figure 2. Student B DT plot for 14 submission until max grade

Figure 3. Submission DT plot from two students for the same exercise

Looking closely to the behavior present in the plot, we noticed that some of the submissions were random modifications or had took place long after the previous submissions. Figure 2 shows how a student with difficulties makes more submissions that do not change their code significantly, potentially indicating that students could be using the automatic evaluation system to guess the right solution. Further analyzing, we identified that most of these submissions have $DT \approx 0.05$. Consequently, we removed all submissions with $DT \leq 0.05$ that did not score max grade from the second transformation step as an attempt to reduce the bias caused by long intervals between submissions and students guessing attempts.

After the removal, we calculated the arithmetic average for $MTES$, $MDES$ and DEX for each exercise. Then, the resulting data were ranked using $MTES$ and $MDES$ values. Submissions with high $MTES$ and $MDES$ respectively were ranked higher than their counterparts. The results presented in the next section are based on the analysis of the resulting average values of $MTES$, $MDES$ and DEX, including other submissions data presented so far.

6. Results & Discussions

When ranking the data we noticed that most students did not answer all exercises. Thus, we tried to find a subset of exercises that maximized the number of students and exercises. This pattern of students not answering all exercises was present in all data set.

After ranking the data based on $MTES$ and $MDES$ scores, respectively, we analyzed the code submitted by students from some of the TOP 10 ranked exercises, in an attempt to identify any pattern. The first data set analyzed was the one associated with iVProg for

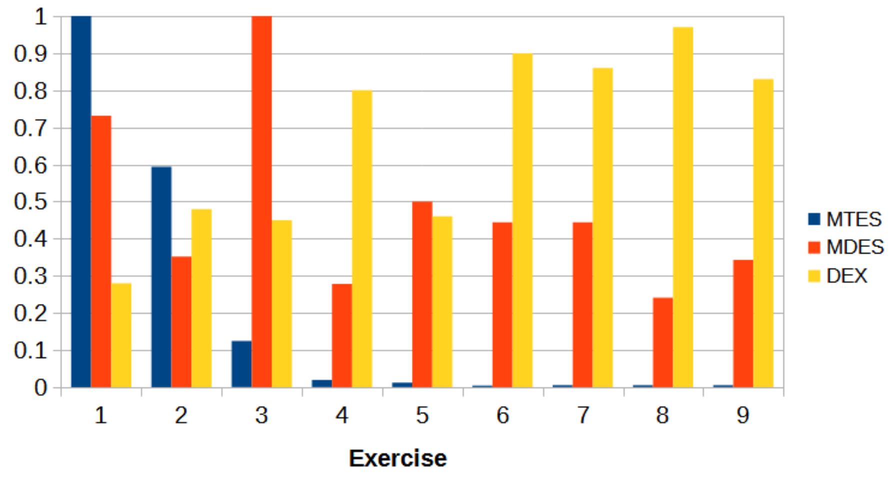

Figure 4. G2 iVProg data $MTES$ and $MDES$ rank

the G2 group, since G1 data set had too few records. G2 iVProg data set had 29 students and 9 exercises. Having more students than G1 and being of a smaller set of exercises when compared to the VPL data, it was easier to notice potential patterns and hypothesis associated with the generated metadata.

Examining the data, we identified that DEX could be associated with exercise difficulty. The ranking of iVProg exercises is presented in Figure 4. $MTES$ and $MDES$ scores were normalized to be inside the interval $[0, 1]$. In the figure, we can notice that the DEX value is getting bigger as we move from left to right. From an specialist point of view, the exercises to the left are more difficult than the ones to the right. As can be noticed in equation (1), we apply a discount in the students' grade based on the time spent to finish the exercise and the number of submissions it took. This is a necessary step because we noticed a binary behavior in students' grade. Since there was no penalty for using the automatic evaluation system, students would try until they found the right solution. In order to account for this behavior, we devised equation (1) to minimize this effect. For example, exercise 1 requires the students to manipulate integers and float in order to split a restaurant bill. Exercise 2 was another challenging for beginners, moving the value from one variable to another.

Another possible relationship present in Figure 4 comes directly from the ranking. Exercises at the left have high $TMES$ and $TDES$ scores, indicating that it could also be associated with exercise difficulties. Since the data set was too small to draw stronger conclusions, we proceeded to analyze the VPL data from groups G1 e G2. This decision was motivated by the fact we had 4 exercises where we asked the students to answer a quick questionnaire before and after finishing them. The questionnaire asked students about what they thought they needed to know to answer it and how difficult it appear to be.

As shown in Figure 5, similar behavior of increasing DEX is also present. The analyzed data consisted of 26 exercises from 17 students. However, if considering only $MTES$ and $MDES$ ranking, the exercises on the left did not correspond to the ones we expected. This time, exercise 1 was a relatively simple problem where the students had to print the name of the month given an integer from 1 to 12. Among the first 5 exercises only

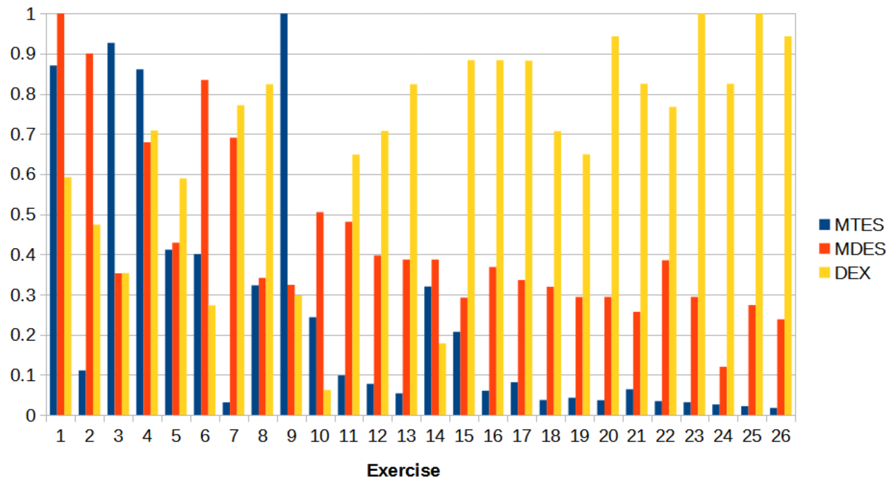

Figure 5. G1 VPL data $MTES$ and $MDES$ rank

exercise 2 could be considered difficult. It required the students to calculate the average of a sequence of numbers until a zero was inputted. Even in the range of the first 10 exercises, very few would be considered difficult, making the assumption about both $MTES$ and $MDES$ implying exercise difficulty not valid. Similar behavior is found on the VPL data from G2, which consists of 25 exercises from 26 students. As opposed to the G1 data set, in this one the first exercise is one considered difficult as it requires the students to use a loop to calculate x^n.

Once it was found that the assumptions about $MTES$ and $MDES$ were not valid, we analyzed the answers to the questionnaire about the exercises difficulties. We ranked the exercises present in the questionnaire and analyzed their associated codes submitted by the students. Additionally, the code solutions for some of the TOP 10 exercises from groups G1 and G2 were also analyzed.

In the G1 data set, only 4 students answered the questionnaire and solved the 4 exercises, while G2 had 6 answers. All questions and exercises were the same for both groups. However, analyzing the comments from the other students, the general feeling toward these exercises is the same. The students were able to come up with some basic understanding of the problems but had difficulty implementing it. It also noticed some problems related to interpreting the exercise description.

By cross-referencing questionnaire data and the ranking of these 4 questions, we identified that $MTES$ and $MDES$ can inform which exercises are challenging the class, not necessarily difficult. These challenges can be related to programming syntax or interpreting the problem description. The interpretation problem can be associated with bad written texts and also students not paying attention to the problem description, resulting in poor understanding and wrong assumptions about the proposed exercise. One of the exercises present in the questionnaire was about print the notes of a music scale from a given note. The music scale was just a context as the question attempt to evaluate the students understanding loops and accessing values in a vector. But, for a student she felt it was necessary to study about music and scales in general in order to understand how to solve the problem. Another possibility for the metrics is that they can likely indicate problems with programming concepts. As anecdotal evidence, the researchers have found students using loop structures as conditional. For example, exercise 1 from Figure 5 students used `switch case` command and had problems with its syntax and usage. This results in high $MTES$ and $MDES$ scores, while maintaining DEX high enough, around 0.6.

Analyzing some of the TOP 10 exercises from G1 and G2, it was possible to confirm that DEX can be used reliably to identify difficult exercises. Even though the DEX score is not the same for both groups for the same exercises, they are relatively close to each other. The standard deviation for each 21 pairs of exercises yield values from 0.0 to 0.21, indicating that they are not that far apart.

Exercises with problems that require the use of loops, nested or chained `if` structures had lower DEX than the others. We also found that high $MDES$ could be associated with syntax problem and could be an indicative of the learning curve for the programming language being used. Exercises with high $MDES$ usually have lots of submissions where students would struggle to properly write some code structure and then give up, changing most of the implementation. The data suggest that, although high $MTES$ score cannot be linked to an specific kind of problem, it can be related to exercises that require some attention regarding their description and complexity. Moreover, when comparing the $MTES$, $MDES$ and DEX scores from a set of 4 common exercises between G1, G2 and G3, it was found that the same patterns were present.

7. Conclusion

This paper presents alternative approaches for data analysis through automatic evaluation tools activities logs. The analysis resulted in three different metrics for the exercises: $MTES$, highest time window between submissions; $MDES$, highest Levenshtein distance between submissions code solutions; and DEX, the average grade according to equation (1). The results shows that the generated metadata has potential to identify problematic exercises. The problems identified are related to language syntax, exercises' textual description and its complexity.

Even though the generated metrics are dependent on the class, they showed similar patterns between different groups. The insights provided by the data can be useful for teachers to mitigate problems with complex exercises as well as descriptions and language syntax. It can also be source of data for automated systems like intelligent tutors which can provide students with tips and also produce alerts to the teachers. However, new experiments must be designed to confirm the properties identified for $MTES$, $MDES$ and DEX scores. More evidences that corroborates the creation of new metrics to identify programming tasks difficulty is still required since the data set was too small to extrapolate the results found. Additionally, the behavior of the metric regarding mandatory classes and non-mandatory ones must be investigated to report potential bias in proposed metrics.Machine Learning models are also included as a next step, not only for easier automation but as a way to gain new insight into the data set.

8. Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

Akçapınar, G., Altun, A., and Aşkar, P. (2019). Using learning analytics to develop early-warning system for at-risk students. *International Journal of Educational Technology in Higher Education*, 16(1):1–20.

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102.

Aleem, A. and Gore, M. M. (2020). Educational data mining methods: A survey. *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 182–188.

Arifi, S. M., Abdellah, I. N., Zahi, A., and Benabbou, R. (2015). Automatic program assessment using static and dynamic analysis. In *2015 Third World Conference on Complex Systems (WCCS)*, pages 1–6.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., and Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599.

Bosse, Y. and Gerosa, M. A. (2015). Reprovações e trancamentos nas disciplinas de introdução à programação da universidade de são paulo: Um estudo preliminar. In *Anais do XXIII Workshop sobre Educação em Computação*, pages 426–435. SBC.

Caspersen, M. E. (2007). *Educating Novices in The Skills of Programming*. PhD thesis, Department of Computer Science.

Chaturvedi, M. (2017). Data mining and it's application in edm domain. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 829–834.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73.

Effenberger, T., Čechák, J., and Pelánek, R. (2019). Measuring difficulty of introductory programming tasks. In *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale*, L@S '19, New York, NY, USA. Association for Computing Machinery.

Felix, I., Souza, L., Brandão, L., Ferreira, B., and Brandão, A. (2019). ivprog: Programação interativa visual e textual na internet. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 8, page 1164.

Félix, I. M., Ambrósio, A. P., Neves, P. S., Siqueira, J., and Brancher, J. D. (2017). Moodle predicta: A data mining tool for student follow up. In *CSEDU (1)*, pages 339–346.

Gomes, A. and Mendes, A. (2007). Learning to program - difficulties and solutions. In *International Conference on Engineering Education*, pages 283–287.

Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, page 86–93, New York, NY, USA. Association for Computing Machinery.

Impey, C. and Formanek, M. (2021). Moocs and 100 days of covid: Enrollment surges in massive open online astronomy classes during the coronavirus pandemic. *Social Sciences Humanities Open*, 4(1):100177.

Lahtinen, E., Ala-Mutka, K., and Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, pages 14–18, New York, NY, USA. ACM.

Liang, C., Majumdar, R., and Ogata, H. (2021). Learning log-based automatic group formation: system design and classroom implementation study. *Research and Practice in Technology Enhanced Learning*, 16(1):1–22.

Lima, M., Carvalho, L., Oliveira, E., Oliveira, D., and Pereira, F. (2020). Classificação de dificuldade de questões de programação com base em métricas de código. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 1323–1332, Porto Alegre, RS, Brasil. SBC.

Milne, I. and Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, 7(1):55–66.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *SIGCSE Bull.*, 39(4):204–223.

Pelánek, R., Effenberger, T., and Čechák, J. (2021). Complexity and difficulty of items in learning systems. *International Journal of Artificial Intelligence in Education*, pages 1–37.

Rodríguez del Pino, J. C., Rubio Royo, E., and Hernández Figueroa, Z. J. (2010). Vpl: laboratorio virtual de programación para moodle. In *XVI Jornadas de Enseñanza Universitaria de la Informática*, pages 429–435. Universidade de Santiago de Compostela. Escola Técnica Superior d'Enxeñaría.

Siemens, G. and Gasevic, D. (2012). Guest editorial-learning and knowledge analytics. *Journal of Educational Technology & Society*, 15(3):1–2.

Sleeman, D. (1986). The challenges of teaching computer programming. *Communications of the ACM*, 29(9):840–841.

Teodoro, L. and Kappel, M. A. (2020). Aplicação de técnicas de aprendizado de máquina para predição de risco de evasão escolar em instituições públicas de ensino superior no brasil. *Revista Brasileira de Informática na Educação*, 28(0):838–863.

Wang, F. L. and Wong, T.-L. (2008). Designing programming exercises with computer assisted instruction. In *International Conference on Hybrid Learning and Education*, pages 283–293. Springer.