Visualizing the difficulty of programming exercises

Jakub Swacha
jakub.swacha@usz.edu.pl

Institute of Management, University of Szczecin,
71-454 Szczecin, Poland

Abstract

Assigning students exercises of adequate difficulty is important for the effective learning of computer programming. The difficulty of programming exercises, though, is not obvious from their description. In this paper, we propose a visual depiction of the difficulty of programming exercises based on the number of attempts each student made to solve it. The proposed visualization is easy to be generated automatically, but also succinct, capable of indicating differences of difficulty of one exercise experienced among various students, and easy to interpret. We illustrate the qualities of the proposed visual depiction using the examples of six programming exercises of varying difficulty, obtained from a submission repository of a real-world programming learning environment.

Keywords: programming education, teaching programming, programming exercises, exercise difficulty, difficulty visualization

1. INTRODUCTION

Learning programming requires constant practice consisting in solving programming exercises (Simões & Queirós, 2020). According to an international survey by Lahtinen at al. (2005), both teachers and students perceive that programming is learned better by self-practice than in classroom lectures.

The programming exercise provided to students should be of adequate difficulty level (i.e., neither too easy nor too difficult) in order to keep students motivated (i.e., neither bored nor frustrated) (Schoeffel et al., 2018). As there are vast differences in the difficulty of programming exercises (Szydłowska et al., 2022), selecting and arranging exercises to meet this requirement is not an easy task, especially as it may happen that exercises considered easy-to-solve by a teacher are later found to be tough-to-pass by students. Obviously, feedback on the difficulty of exercises is needed by a teacher to adjust the selection and order of exercises provided to students. Programming learning environments usually provide such feedback in a limited, textual form, comprising only basic data, such as the number of correct answers, the ratio of correct answers, or the number of hint requests (see, e.g., Feng & Heffernan, 2005). In this paper, we argue, that information on the difficulty of exercises can be more effectively conveyed in the visual form, which not only allows more detailed data to be presented without sacrificing succinctness, but is also simple to generate automatically and easy to interpret.

Our proposal is described in section 3, whereas in section 4, we provide real-world examples of programming exercise difficulty

visualization to illustrate its qualities. Before that, in the following section, we discuss selected prior work on programming exercise difficulty.

2. RELATED WORK

Although our search for "programming exercise difficulty visualization" yielded no results on Google Scholar, suggesting the approach proposed here is novel, the problem of programming exercise difficulty has been addressed by prior research, focusing, however, on aspects other than visualization.

Effenberger et al. (2019) dedicate their work to the problems of measuring difficulty and complexity of introductory programming problems. They investigated two measures: lines of code and the number of concepts, and empirically found out that neither is an accurate predictor of difficulty of a programming exercise.

Nguyen et al. (2021) investigated the same problem on the example of an introductory data science course and a set of four metrics: Halstead Volume, Cyclomatic Complexity, number of library calls and number of common library calls. They reported that these metrics could identify cases where students submitted overly complicated codes and therefore would benefit from providing a scaffolding. Moreover, they especially pointed to the number of library calls, as a significant predictor of the change in submission score.

Tirronen and Tirronen (2020) applied the Performance Factors Analysis model to a pass/fail data coming from an introductory programming course. Although they found out that their approach did not provide credible information on student outcomes on the course, it fared well in estimating the exercise difficulty.

Kalemba and Ade-Ibijola (2019) propose a tool called NOPCE (The NOvice Program Complexity Estimator) that measures the complexity of programming problems based on the constructs their C# solutions contain, employing a look-up table containing predefined weights for programming constructs. They report that humans agreed with NOPCE on problem ranking for most of the time.

Intisar and Watanobe (2018) who emphasize the importance of providing exercises matching students' experience and level, propose an expert system capable of categorizing the programming problems based on their difficulty using fuzzy rules derived by performing cluster analysis on submission log data.

Skarbalius and Lukoševičius (2021) also address the problem of evaluating the difficulty of programming exercises yet use a different kind of input data for this purpose: text description of the problem with accompanying figures.

Craig et al. (2017) investigate an interesting aspect of the programming exercise difficulty problem, that is whether problems stated in a context familiar to students are easier to solve than the same problems stated in an unfamiliar domain. Their results suggest that any advantage given by a familiar context is dominated by other factors, such as the complexity of terminology used in the problem description, its length, and the availability of examples, which suggests that exercise authors should focus on the simplicity of language and the development of examples, rather than putting the problems in contexts familiar to students.

Tiam-Lee and Sumi (2018) deal with the more complex problem of procedurally generating programming exercises, and propose a system for this purpose, which is, however, capable also of adjusting the exercise difficulty.

None of the works described above take the visual approach as the one proposed in the following section.

3. VISUAL REPRESENTATION OF PROGRAMMING EXERCISE DIFFICULTY

In programming learning environments, the feedback on the difficulty of solving programming exercises by students is provided usually as one or few numbers – e.g., the number of correct answers, the ratio of correct answers, or the number of hint requests (see, e.g., Feng & Heffernan, 2005). This does not give a proper picture of the exercise difficulty, as difficulty is subjective, and even one exercise may be experienced by various students in different ways. Therefore, in order to depict that, one needs to see not only just one aggregate number, but how the exercise difficulty was experienced by all students who attempted to solve it. To accomplish that, we propose to use a graphical summary of the number of each student's attempts on solving an exercise in the form of a bar chart.

One such chart is constructed for each exercise, and each bar in such a chart depicts the number of one student's attempts on solving that exercise. Only the attempts by students who eventually solved the exercise are considered. Therefore, usually the harder the exercise, the less bars there will be, as fewer students managed to solve it.

To increase the readability of the chart, the bars in it are ordered ascendingly according to their length, and the Y axis is scaled accordingly to the tallest bar in the chart – although a scale based on the tallest bar in a chart collection could alternatively be used if the purpose is to visually compare the difficulty of various exercises.

An easier exercise will be solved with fewer attempts than a difficult one. By capturing the number of attempts of all students, the chart is able to indicate a difference between an exercise which is of medium difficulty for all students (a flat chart with bars of similar height) and one which is easy for some, yet difficult for others (a strongly skewed chart), which could not be shown with just a single aggregate.

If other variables are to be taken into consideration, e.g., whether a student asked for hints before solving an exercise, a stacked bar chart could be used.

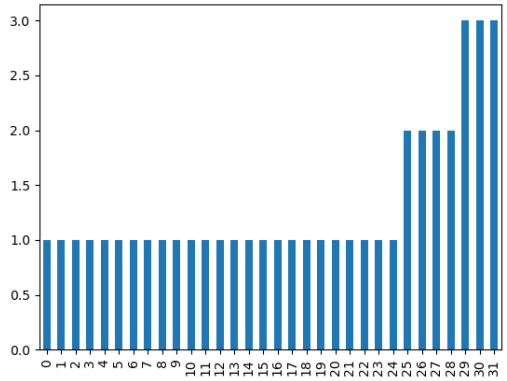
4. PRACTICAL IMPLEMENTATION AND EVALUATION

A proof-of-concept generator of charts of programming exercise difficulty has been implemented in Python using pandas and matplotlib libraries.

The chart generation has been tested on the "Introduction to Python 3 programming" exercise set containing 94 exercises for which over 9000 student solution attempts have been recorded (Szydłowska et al., 2022). All the data were obtained from an instance of the FGPE Gamified Programming Learning Environment (Paiva et al., 2021).

Figures 1-6 demonstrate the ability of the proposed visualization method to indicate differences in difficulty of programming exercises. In Fig. 1, we can see a chart generated for a very easy exercise, as most students solved it in the very first attempt, and only few of them needed two or three attempts.

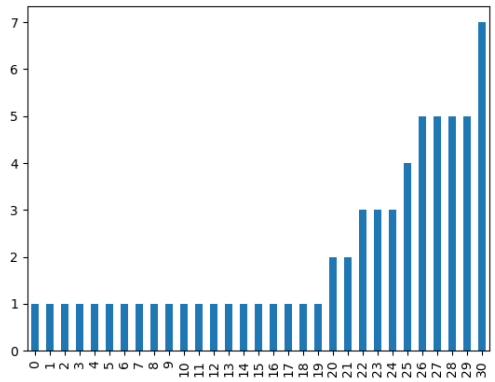
Figure 1: *Visualization of a very easy exercise*



The exercise depicted in Fig. 2 is also easy, yet less than the previous one: almost half

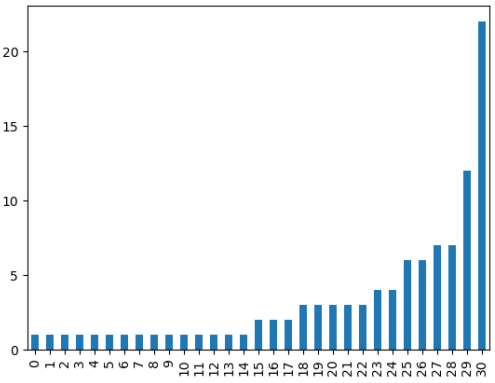
of students needed at least two attempts to solve it and one student even needed 7 attempts.

Figure 2: *Visualization of an easy exercise*



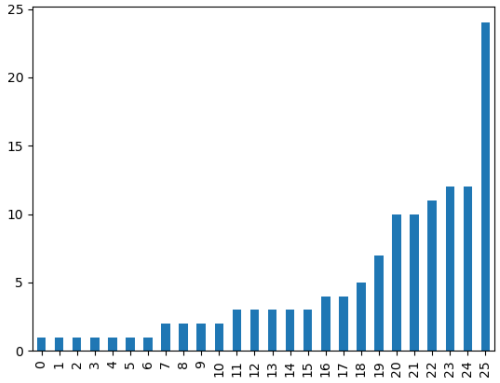
The exercise depicted in Fig. 3 is of mixed character: it was still easy for most students, yet 6 students needed at least 6 attempts to solve it, and one student needed over 20 attempts.

Figure 3: *Visualization of an easy exercise with an outlier*



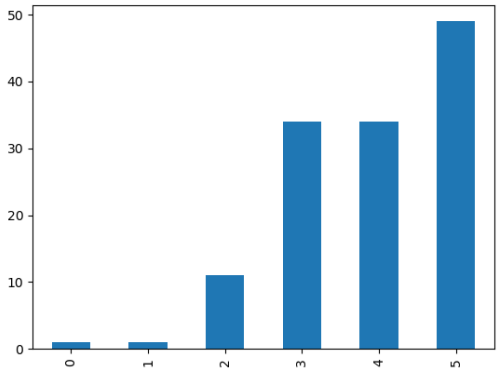
In Fig. 4, an exercise of medium difficulty has been portrayed. For about one-third of the students one or two attempts were enough to have it solved, another one-third took less than 5 attempts, and the last one-third of the students took 10 or more attempts to solve it.

Figure 4: *Visualization of a medium difficulty exercise*



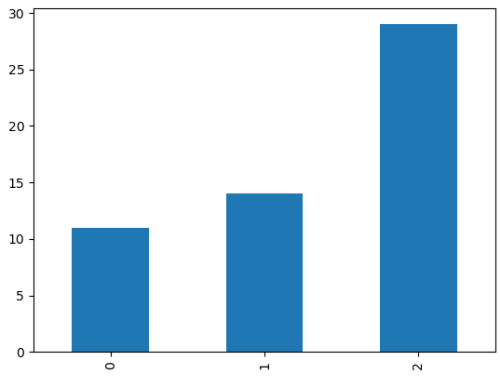
The exercise shown in Fig. 5 can be considered difficult, as only 6 students have solved it; two of them made it in their first attempt, the others took at least 10 attempts (the last student needed almost 50 attempts to solve the exercise).

Figure 5: *Visualization of a difficult exercise*



Finally, in Fig. 6, a very difficult exercise is charted: there were only three students who solved it, and for no student it took less than 10 attempts.

Figure 6: *Visualization of a very difficult exercise*



The charts presented above confirm that the proposed method for visualization of programming exercise difficulty is capable of indicating differences among the exercises in an easy-to-interpret way.

5. CONCLUSION

In this paper, we have shown that the difficulty of programming exercises can be depicted visually, providing the teachers with an easy-to-interpret information on how difficult a given exercise was for the students. What is important, the generated charts convey detailed information with regard to all students (not just an average), what, e.g., allows to identify exercises which, although easy for most students, are very difficult for some of them.

A proof-of-concept implementation of the visualization method has been implemented in Python and successfully tested on real-world exercise data obtained from a programming learning environment.

A possible future work is to embed the exercise difficulty chart generator in a programming learning environment and then perform its usability evaluation among the teachers using that programming learning environment.

6. REFERENCES

Craig, M., Smith, J., & Petersen, A. (2017). Familiar Contexts and the Difficulty of Programming Problems. In Proceedings of the 17th Koli Calling International Conference on Computing Education Research, 123–127. https://doi.org/10.1145/3141880.3141898.

Effenberger, T., Čechák, J., & Pelánek, R. (2019). Difficulty and Complexity of Introductory Programming Problems. In Educational Data Mining in Computer Science Education Workshop. https://www.fi.muni.cz/~xpelanek/publications/difficulty-complexity-programming.pdf.

Feng, M., & Heffernan, N. T. (2005). Informing Teachers Live about Student Learning: Reporting in Assistment System. In Workshop on Usage Analysis in Learning Systems, Proceedings of Artificial Intelligence in Education. http://lium-dpuls.iut-laval.univ-lemans.fr/aied-ws/PDFFiles/feng.pdf.

Intisar, C. M., & Watanobe, Y. (2018). Cluster Analysis to Estimate the Difficulty of Programming Problems. In Proceedings of the 3rd International Conference on Applications in Information Technology, 23–28. https://doi.org/10.1145/3274856.3274862.

Kalemba, E., & Ade-Ibijola, A. (2019). A Metric for Estimating the Difficulty of Programming Problems by Ranking the Constructs in their Solutions. In 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC), 1–9. https://doi.org/10.1109/IMITEC45504.2019.9015843.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. ACM SIGCSE Bulletin, 37(3), 14–18.

Nguyen, H., Lim, M., Moore, S., Nyberg, E., Sakr, M., & Stamper, J. (2021). Exploring Metrics for the Analysis of Code Submissions

in an Introductory Data Science Course. In LAK21: 11th International Learning Analytics and Knowledge Conference, 632–638. https://doi.org/10.1145/3448139.3448209.

Paiva, J. C., Queirós, R., Leal, J. P., Swacha, J., & Miernik, F. (2021). An Open-Source Gamified Programming Learning Environment. In P. R. Henriques, F. Portela, R. Queirós, & A. Simões (Eds.), Second International Computer Programming Education Conference (ICPEC 2021) (Vol. 91, p. 5:1-5:8). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.ICPEC.2021.5.

Schoeffel, P., Wazlawick, R. S., & Ramos, V. F. C. (2018). Motivation and Engagement Factors of Undergraduate Students in Computing: A systematic mapping study. In 2018 IEEE Frontiers in Education Conference (FIE), 1–5. https://doi.org/10.1109/FIE.2018.8658384.

Simões, A., & Queirós, R. (2020). On the Nature of Programming Exercises. In R. Queirós, F. Portela, M. Pinto, & A. Simões (Eds.), First International Computer Programming Education Conference (ICPEC 2020) (Vol. 81, p. 24:1-24:9). Schloss Dagstuhl–Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.ICPEC.2020.24.

Skarbalius, A., & Lukoševičius, M. (2021). Automatic Programming Problem Difficulty Evaluation – First Results. In A. Lopata, D. Gudonienė, & R. Butkienė (Eds.), Information and Software Technologies (pp. 150–159). Springer International Publishing. https://doi.org/10.1007/978-3-030-88304-1_12.

Szydłowska, J., Miernik, F., Ignasiak, M. S., & Swacha, J. (2022). Python Programming Topics That Pose a Challenge for Students. In A. Simões & J. C. Silva (Eds.), Third International Computer Programming Education Conference (ICPEC 2022) (Vol. 102, p. 7:1-7:9). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/OASIcs.ICPEC.2022.7.

Tiam-Lee, T. J., & Sumi, K. (2018). Procedural Generation of Programming Exercises with Guides Based on the Student's Emotion. In 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 1465–1470. https://doi.org/10.1109/SMC.2018.00255.

Tirronen, V., & Tirronen, M. (2020). Estimating Programming Exercise Difficulty using Performance Factors Analysis. In 2020 IEEE Frontiers in Education Conference (FIE), 1–5. https://doi.org/10.1109/FIE44824.2020.9274142.