Measuring Difficulty of Introductory Programming Tasks

Tomáš Effenberger
Masaryk University
Brno, Czech Republic
tomas.effenberger@mail.muni.cz

Jaroslav Čechák
Masaryk University
Brno, Czech Republic
410322@mail.muni.cz

Radek Pelánek
Masaryk University
Brno, Czech Republic
pelanek@fi.muni.cz

ABSTRACT

Quantification of the difficulty of problem solving tasks has many applications in the development of adaptive learning systems, e.g., task sequencing, student modeling, and insight for content authors. There are, however, many potential conceptualizations and measures of problem difficulty and the computation of difficulty measures is influenced by biases in data collection. In this work, we explore difficulty measures for introductory programming tasks. The results provide insight into non-trivial behavior of even simple difficulty measures.

INTRODUCTION

The central question that we address in this work is "How should we measure the difficulty of problem solving tasks in learning systems?" We are concerned specifically with adaptive learning systems, in which students solve tasks and the system collects data on their performance. For our analysis, we use data about introductory programming tasks, but the studied issues are directly relevant also for other problem solving tasks, e.g., in mathematics or physics.

Measuring the difficulty of tasks in learning systems is useful from several perspectives. Data on student performance are used by student modeling techniques to estimate skills of students [10]. The obtained skill estimates are used to guide adaptive behavior of systems, e.g., for mastery criteria or recommendations. By taking the difficulty of tasks into account, we can improve the estimates [9, 6] and thus the behavior of systems. A specific important application of task difficulty measures is for task ordering. A common approach in learning systems (both adaptive and non-adaptive) is to sequence tasks in the order from easier to more difficult [1]. In order to do so, we need to have a good measure of difficulty. Difficulty measures are also useful for content management, i.e., for creation, removal, and updates of tasks, and for domain modeling. Information about difficulty is useful feedback to task creators and to authors of domain models.

Quantifying the difficulty of tasks is, however, quite non-trivial. On a conceptual level, it is not even clear what the concept of "difficulty" means. For example, many authors discuss the distinction between "complexity" and "difficulty" and their exact meanings [12, 2, 8]. Although the exact views of these concepts differ among authors, the common view is that complexity is concerned with intrinsic properties of a task itself (structure, components, their relations), whereas difficulty is a matter of person-task interaction. Measuring complexity is thus to a large degree specific for a particular type of problem solving exercise, whereas difficulty can be measured using performance data, which have a very similar form for different types of exercises. We thus focus on measures of difficulty, which are more readily applicable in a wide range of learning systems.

Even when we take the view of difficulty as a measurement of solvers' performance, it is far from clear what aspects of performance we should consider and how to quantify difficulty. For example, two difficulty dimensions of performance that are available in most contexts are time intensity (measured for example as median response time for correct answer) and failure rate (ratio of students unable to finish the task). In many cases, additional data can be used to quantify difficulty, e.g., "process data" like the number of steps or the quality of solutions. Each of these aspects concerns different facet of difficulty, and often they can be quantified in several ways, i.e., we can obtain many different measures of difficulty. However, for the practical development of learning systems, it is very useful to quantify the difficulty of tasks by a single number. Which one should we choose?

Once we solve the conceptual issues and decide which specific facet of difficulty we want to measure and what specific measure we want to use (e.g., median problem solving time), we still have to face methodological issues with data collection [11]. Observed performance data from learning systems are influenced by biases. Performance of students on a particular task depends on the context, specifically ordering of tasks. Was the task preceded by a very similar task (and thus the performance is influenced by transfer)? Is the task the first one in a problem solving sequence and thus the student is getting acquainted with the user interface? Data from learning systems also often include attrition bias—students leave the system in non-random order and thus populations subsamples for different tasks differ. These effects can potentially have a significant influence on the computed difficulty measures.

In this work, we analyze data from learning systems for several types of introductory programming exercises. We explore relations among different approaches to measuring problem difficulty, stability of measures, and effects of data filtering.

L@S '19 June 24–25, 2019, Chicago, IL,USA

ACM ISBN 978-1-4503-6804-9/19/06.
DOI: https://doi.org/10.1145/3330430.3333641

ANALYSIS OF DIFFICULTY MEASURES

We use data from two systems for the practice of introductory programming (`robomise.cz` and `umimeprogramovat.cz`) and analyze the properties of basic difficulty measures like failure rate and median problem solving time.

Data

Table 1 presents an overview of four programming exercises that we use to explore the relationship between different methods for measuring task difficulty. These exercises practice basic programming concepts like sequences of commands, conditional statements, loops, variables, and functions; however, not all the exercises practice all of these concepts. Students can execute their program at any time to see its output. For the analysis, we use summary information about attempts: success (whether the student eventually solved the task), solving time, number of edits, and number of executions.

In the RoboMission exercise [5], students build programs using a block-based programming interface to guide a robot on a grid. The tasks are divided into levels and sublevels. After completing a task, the system recommends to the student a randomly chosen task from the first sublevel which has not been yet mastered. The other three exercises come from another learning system (`umimeprogramovat.cz`). Each of them contains 5–10 linearly ordered levels and the tasks are recommended in a predefined order. These three exercises use scaffolding in the form of prepared programs that need to be modified or extended. In both Turtle Blockly and Turtle Python, students command a turtle to draw assigned geometric objects [3]; the difference between the two is in the programming interface (block-based vs. textual). In the fourth exercise, students write any Python code to solve tasks with numbers, strings, and lists.

Table 1. Programming exercises and data used for analysis.

Exercise	Interface	Tasks	Students	Attempts
RoboMission	blocks	85	3,800	62,500
Turtle Blockly	blocks	77	11,000	63,600
Turtle Python	text	51	2,400	11,900
Python	text	73	2,000	10,700

Relations between Performance Aspects

Many aspects of student performance can be used for measuring the difficulty of tasks. In a programming exercise, the learning system can collect granular data on students' problem solving activity, i.e., a timed sequence of individual submits, code edits, or even keystrokes [7]. For our analysis, we take two basic aspects of performance: the failure rate (ratio of students who started solving a task but did not finish it) and the median time of correct solutions. These measures are the most universal—they are not dependent on specific aspects of data collection and thus can be used in virtually any system for introductory programming (or problem solving in general).

Fig. 1 shows the relation between these two measures for our four introductory programming exercises. We see that in all cases the correlation between the measures is not perfect, i.e., each measure captures a different aspect of difficulty. The degree of correlation moreover differs among the exercises. In RoboMission, the measures are reasonably well correlated, whereas for Python programming, the correlation is much lower. These results show that even for quite similar types of exercises, it is not possible to make general conclusions about the relation of different difficulty measures.

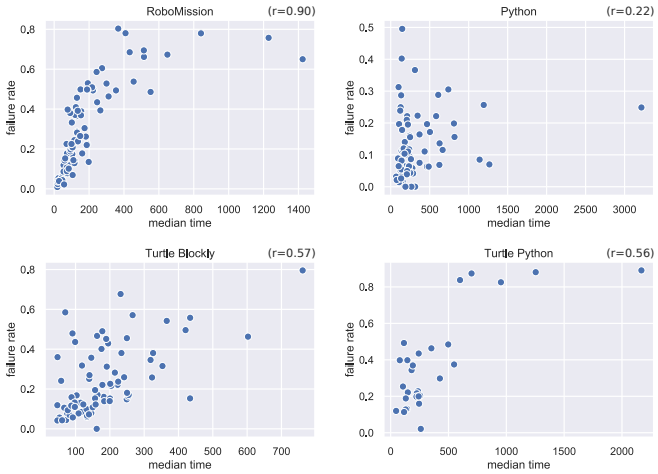

Figure 1. Comparison of a measure relation across different exercises. Next to each exercise name there is a Spearman correlation coefficient.

Difficulty and Complexity

The complexity of a task depends only on its content and thus can be measured (at least approximately) before any performance data are collected. The complexity of programming tasks is influenced by factors such as required programming concepts (e.g., conditional statements, while loops, and nesting flow-of-control structures), the total number of flow-of-control structures, and total number of lines in the solution. Fig. 2 shows the relationship between estimated complexity and observed difficulty for tasks in RoboMission. Complexity is estimated as the average of 3 ranks according to (1) the number of distinct programming concepts, (2) the total number of flow-of-control structures, and (3) the total number of lines in the solution. Difficulty is estimated as the average of 4 ranks according to (1) the failure rate, (2) the median solving time, (3) the median number of edits, and (4) the median number of executions. The correlation between complexity and difficulty is moderate (Spearman's $r = 0.73$), suggesting that complexity can serve as a reasonable proxy for the difficulty of a new task.

A significant discrepancy between difficulty and complexity indicates a potentially problematic or wrongly placed task. In RoboMission, most of the tasks that are more complex than difficult have quite a straightforward solution and include only concepts that are already practiced in previous levels. These tasks might be more appropriate in an earlier level, in which one of the concepts they include was introduced for the first time.

On the other end of the spectrum, there are a few tasks in RoboMission that are considerably more difficult than complex. Most of them are tricky tasks that do not require any advanced programming concept, but it is difficult to find the correct sequence of actions to solve the puzzle. The trickiness

could be argued to be a part of the complexity; however, we cannot capture it by simple indicators using only task content. Comparing the estimated complexity with the observed difficulty helps to detect and flag these tricky tasks, and such information could be used for task recommendation. For example, a learning system should avoid giving tricky tasks to struggling students, but can give them to more experienced students to make their progress through introductory levels more challenging.

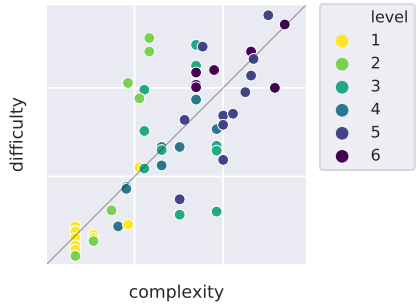

Figure 2. Comparison of complexity and difficulty of tasks in RoboMission.

Stability of Measures

Computing task difficulty requires that enough performance data about the task has already been collected. Generally, a few hundreds of attempts are needed to obtain difficulties that result in a reasonably stable task ordering, but the rate of convergence differs across difficulty measures. Fig. 3 shows how the number of students impacts the stability of induced task orderings for individual difficulty measures. The final task ordering is computed from a randomly sampled half of the students; the other half of the students is then used to create samples of increasing size. The stability is computed as the average Spearman correlation coefficient between the final ordering and orderings induced from random samples of given number of students.

The lowest number of students is required by measures based on the number of edits and solving time and the highest number of students by the failure rate. This is in line with the intuition that the more bits of information from the students' attempts is used, the faster the convergence to a stable ordering. Combining multiple performance data might thus result in even more stable ordering; however, in RoboMission, a straightforward combination of the individual measures (as a weighted average of ranks) is less stable than just using the number of edits or the solving time alone.

Biases and Filtering

Collected data are influenced by various biases. Many students in our systems attempt to solve just a single task. These students are just looking around and they are not seriously solving tasks. Such behavior increases the failure rate of some tasks. Moreover, students solving their first few tasks are often struggling with user interface rather than with the content of tasks. If students encountered the very same task later, they might be able to solve it easily. This condition increases the

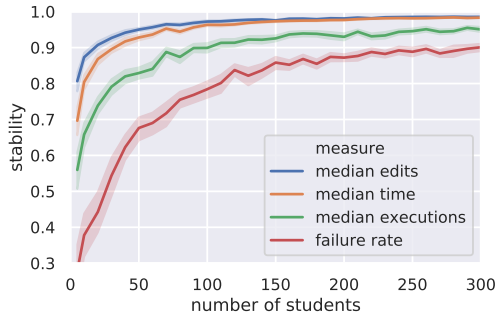

Figure 3. Stability of individual difficulty measures computed as Spearman correlation coefficient between a running and the final task orderings. The shaded regions correspond to 95% confidence intervals.

solving time of the tasks encountered early in the solving sequence. We explore the impact of these biases in several programming exercises.

The desired property of difficulty measures is to be insensitive to the mentioned biases. To asses the insensitivity we compute the measure on a sub-sequence of tasks attempted by each student. The sub-sequence starts with k-th attempted task. This filtering removes both non-serious attempts and attempts where the student is learning how to use the system. The downside is a significant reduction of data and thus the decrease of stability of measures.

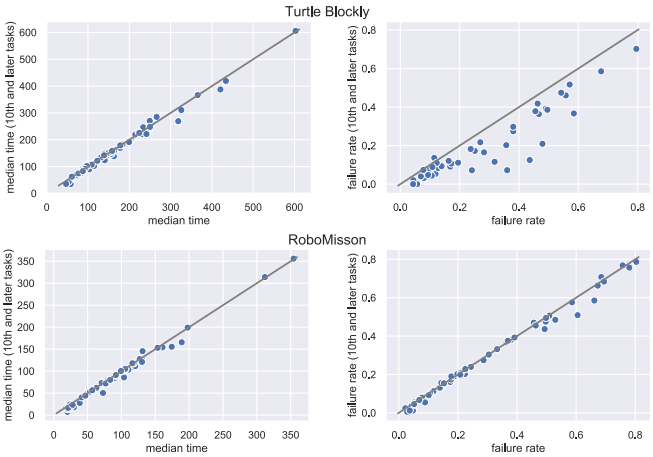

Figure 4. Comparison of failure rate and median solving time measures with and without filtering of the first items from the solving sequence. The top row uses Turtle Blockly data and the bottom row uses RoboMission data.

We studied the effects of filtering on failure rate and median solving time measures. Fig. 4 shows a correlation of measure computed on complete and filtered data. The filtering is done by taking only sub-sequences starting from the 10th attempted task by each student. A high correlation, in this case, means that the measure produces similar results both on the complete and the filtered data. Since the first attempts have the most potential to be biased, high correlation suggests insensitivity of the measure to mentioned biases.

The correlations differ between measures and exercises. The filtering has minimal impact on median solving time in all studied exercise types. The failure rate, on the other hand, tends to decrease on the filtered data. The significance of the effect depends on a particular exercise. Fig. 4 shows data from two similar exercises (both based on the Blockly programming environment). Nevertheless, the two exercises are from learning systems with a different approach to task sequencing. RoboMission uses an adaptive recommendation algorithm that includes randomization of ordering among tasks with similar difficulty. Consequently, the biases caused by problem ordering are probably limited and the filtering of data has minimal impact on failure rate. Other studied exercises are from a system with the fixed ordering of tasks. In these cases, the filtering has a nontrivial effect, particularly on tasks that occur on the first position in one of the predefined task sets.

DISCUSSION AND FUTURE WORK

If task difficulties are used to inform decisions, such as which task to recommend or how to order the tasks, the measurement of difficulty requires careful attention. We have demonstrated that in different programming exercises, decisions about the choice of difficulty measure and bias-reducing filtering can have considerably different effects on the final measurements and induced ordering of tasks. Furthermore, the stability of the measurements can be low if there are not enough performance data collected. In examined programming exercises, the median solving time measure converges faster than the failure rate measure. The median time measure is also more robust with respect to biases caused by task ordering.

Computation of difficulty suffers from the cold start problem; it requires a few hundreds of students that have solved the task before the difficulty estimates are stable. Before the performance data is collected, a complexity computed from the task content can be used as a reasonable approximation of the difficulty. Future research could explore how to reduce the required amount of collected data by combining several difficulty and complexity measures, as well as other available information. Frequently, new tasks are added to an existing exercise; in such case, the collected data about older tasks, together with a suitable task similarity measure could be used to obtain difficulty estimate for the new task.

In this work, we have examined simple measures of difficulty and a method to reduce biases through straightforward filtering of attempts. A natural extension is to employ models from item response theory [4], which have the group invariance property (the difficulty estimates are not dependent on the subpopulation that solves a task). However, these models do not take into account learning, so they would also not provide an optimal solution. Moreover, robustly estimating completely context-independent difficulty might be infeasible if the learning system encourages the students to proceed through the tasks in a similar order [11]. As an alternative approach, we suggest to explore estimation of *relative contextual difficulty*; i.e., whether the task is of appropriate difficulty given its placement within the system, or whether it is too easy or too difficult. Such information would be helpful for small iterative improvements of the tasks and their division into levels.

REFERENCES

1. Peter L Brusilovsky. 1992. A framework for intelligent knowledge sequencing and task sequencing. In *International Conference on Intelligent Tutoring Systems*. Springer, 499–506.
2. Donald J Campbell. 1988. Task complexity: A review and analysis. *Academy of management review* 13, 1 (1988), 40–52.
3. Michael E Caspersen and Henrik Bærbak Christensen. 2000. Here, there and everywhere - on the recurring use of turtle graphics in CS 1. In *ACM International Conference Proceeding Series*, Vol. 8. 34–40.
4. R.J. De Ayala. 2008. *The theory and practice of item response theory*. The Guilford Press.
5. Tomáš Effenberger and Radek Pelánek. 2018. Towards making block-based programming activities adaptive. In *Proc. of Learning at Scale*. ACM, 13.
6. JP González-Brenes, Yun Huang, and Peter Brusilovsky. 2014. General features in knowledge tracing: Applications to multiple subskills, temporal item response theory, and expert knowledge. In *Proc. of Educational Data Mining*. 84–91.
7. Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, and others. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 41–63.
8. Peng Liu and Zhizhong Li. 2012. Task complexity: A review and conceptualization framework. *International Journal of Industrial Ergonomics* 42, 6 (2012), 553–568.
9. Zachary A Pardos and Neil T Heffernan. 2011. KT-IDEM: introducing item difficulty to the knowledge tracing model. In *International conference on user modeling, adaptation, and personalization*. Springer, 243–254.
10. Radek Pelánek. 2017. Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques. *User Modeling and User-Adapted Interaction* 27, 3 (2017), 313–350.
11. Radek Pelánek. 2018. The details matter: methodological nuances in the evaluation of student models. *User Modeling and User-Adapted Interaction* (2018).
12. Judy Sheard, Angela Carbone, Donald Chinn, Tony Clear, Malcolm Corney, Daryl D'Souza, Joel Fenwick, James Harland, Mikko-Jussi Laakso, Donna Teague, and others. 2013. How difficult are exams?: a framework for assessing the complexity of introductory programming exams. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*. Australian Computer Society, Inc., 145–154.