Masaryk University
Faculty of Informatics

Educational Data Analysis for Introductory Programming

Master's Thesis

Matěj Vaněk

Brno, Fall 2018

Masaryk University
Faculty of Informatics

Educational Data Analysis for Introductory Programming

Master's Thesis

Matěj Vaněk

Brno, Fall 2018

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matěj Vaněk

Advisor: doc. Mgr. Radek Pelánek, Ph.D.

Acknowledgements

I would like to thank my supervisor, doc. Mgr. Radek Pelánek, Ph.D., for his vast knowledge and experience, valuable remarks and the entire lead of this thesis, Mgr. Tomáš Effenberger for his willingness and friendliness while dealing with my questions and requirements to his system, and least but not last my parents and my fiancée Zuzka for all the kinds of so-necessary support which I have received.

Abstract

This thesis presents an analysis of possible ways how to measure characteristics of tasks and learners in the field of introductory programming e-learning systems. Particular analyses are conducted on data from the system RoboMission. Results of these analyses are presented in the form of a dashboard intended for system's developers.

Keywords

introductory programming, e-learning, correlation analysis, dashboard, RoboMission

Contents

1 Introduction

E-learning is a modern, developing way how to approach to education using the opportunities of computers and the Internet. These technologies enable teachers and creators of e-learning systems to transmit desired knowledge to much more learners than it would be possible in traditional school lessons. The ease and relative effectiveness of e-learning systems caused their significant development in manyfields in the last two decades. Programming (and computer science in general) is not an exception.

While maintaining an e-learning system, every administrator of educational content needs proper information about the performance and characteristics of educational items (tasks) and users of the system (learners). Due to the fragmented specialization of e-learning systems domains, the variety of possible learning mechanisms, and the novelty of the e-learning as a subject of science, processes how to obtain this needed knowledge are not standardized.

To properly specify the information which we are interested in, we define the following seven research questions – How can we measure:

1. the difficulty of a task?
2. the complexity of a task?[1]
3. the uniqueness of a correct solution of a task?
4. the similarity of a task to some other task?
5. which problems do learners have while solving a task?
6. a learner's performance on a task?
7. a learner's performance in the whole system?

This thesis aims to find ways how to properly obtain and express this information about tasks and learners in the field of introductory programming e-learning systems. We use the e-learning system RoboMission from Masaryk University as a base for our research; all

1. The difference between difficulty and complexity is presented in Section 4.1.

the following analyses are performed on its data. Results of this research are summarized into the form of a dashboard which is intended for once-in-a-while use by RoboMission developers and teachers (who may use this system in their school classes in future and control the progress of their students).

The motivation for this research is a better understanding of the properties of adaptive e-learning systems in the field of programming. The advance in personalized learning is one of the *14 Grand Challenges for Engineering in the 21st Century* announced by US National Academy of Engineering [1]. Personalized e-learning is one of the ways how to realize this advance on a great scale, projects like Hour of Code[2] teach millions of users worldwide to program. The impact of these activities brings the importance of their research.

We study properties of RoboMission tasks and learners and create a solid base for improvement of the entire system. Our analytic process is transferable to other introductory programming e-learning systems where it can be applied with respect to the systems' specific features.

The structure of this thesis is as follows: In Chapter 2 we describe the field of the introductory programming e-learning systems, measures used for a description of phenomena in systems of this type, and the visualization of data in the form of dashboards. Chapter 3 is concerned with our source system RoboMission, description of our data, and the procedure of data processing. The task characteristics are discussed in Chapter 4, the learner characteristics in Chapter 5. Our resulting dashboard is presented in Chapter 6.

2. Website https://www.code.org/ .

2 Related Work

This chapter describes our research in the fields of introductory programming e-learning systems (Section 2.1), measures which these systems use for the description of tasks and learners (Section 2.2), and an overview of existing visualizations of measures which are considered informative (Section 2.3).

2.1 Introductory Programming E-learning Systems

Introductory programming learning is concerned with the very first moments when a learner meets programming. Its main goal is to make a learner familiar with the basic principles of programming languages and algorithmic thinking. These principles include constructs such as commands, loops, conditions, variables, functions, and recursion.

The introductory programming e-learning systems can be divided according to two criteria.

The first criterion is the form in which a program is constructed. Two main code-construction ways are the textual and the block ones. The textual construction consists of program-code writing character-by-character. Here, a learner has to learn both semantic and syntactic aspects of the code structure at once. The block construction approach is based on the program composition from prepared blocks. A block represents one code element such as command or control statement, often with slots which demonstrate syntactic relations between blocks (e. g. `"if"` block has slots for a boolean expression and body blocks). This facilitation leads to a restricted domain of possible programs and avoidance of syntactic errors; learners learn the program syntax only implicitly. Drag-and-drop block principle is also more friendly to mobile devices users who may be uncomfortable with typing on a keyboard or a display.

The second dividing criterion is the environment in which a program output is processed. Classical console environment used in the programming practice is very minimalist – based on the program and input, it returns only a textual response. This is very effective, but on the other hand, beginners who are used to intuitive graphical interfaces may consider this unusual and less user-friendly. Unlike the

console environment, environments with graphical output present the program output in a context of the system-specific game world. The most frequent game worlds are the drawing and the grid ones. In the drawing game world, a learner controls a pen (often represented by a turtle's tail) by series of distance and direction-change commands. The grid game world is composed of squares forming a grid on which a robot walks and performs actions.

This systems division can be demonstrated in the existing introductory programming e-learning systems. Figures 2.1 and 2.2 present four systems of different types. *Hour of Code: Code with Anna and Elsa*[1] is a representative of block-based drawing environments, *Robotanist*[2] and *RoboMission*[3] represent block-based grid world systems, and *Interactive Python*[4] is an instance of console environments. The last three systems were developed by Adaptive Learning research lab at Faculty of Informatics, Masaryk University[5].

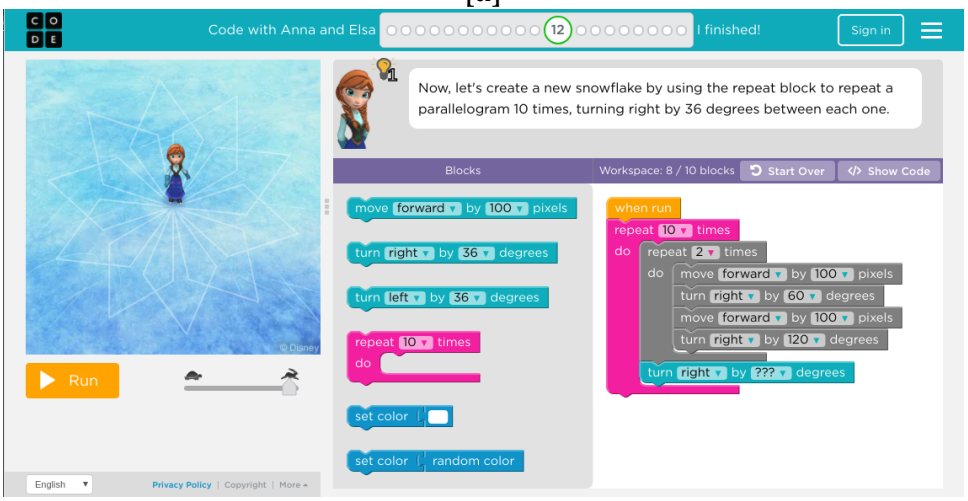
2.2 Task and Learner Measures

In order to obtain information about characteristics used for a description of tasks and learners in the field of introductory programming e-learning systems, research of literature was performed. We found 15 articles from the last 11 years which are at least partially concerned with this topic. We found 41 data measures and characteristics mentioned in the articles, they can be seen in tables 2.1, 2.2 and 2.3. This set comes from e-learning systems with various types of content and logging, and therefore only a part of these items is at least theoretically usable in the context of RoboMission; the usable ones are marked as "relevant".

The measures are divided into ten categories based on its focus. Approximately one-half of categories is widely represented in the literature (time, edits, errors, score, task similarity) while the others contain only a few measures from a small number of papers. Also,

1. Website https://studio.code.org/s/frozen/ .
2. Website https://www.umimeprogramovat.cz/robotanik , only in Czech.
3. Website https://en.robomise.cz .
4. Website https://www.umimeprogramovat.cz/programovani-v-pythonu , only in Czech.
5. Website https://www.fi.muni.cz/adaptivelearning/ .

[a]



[b]

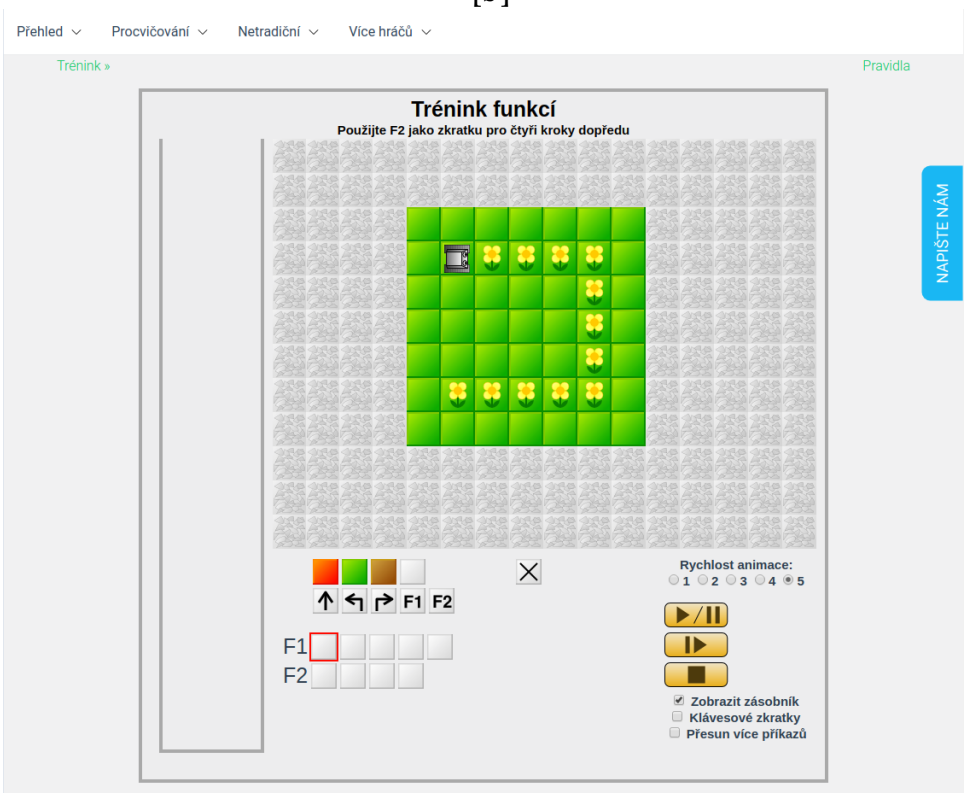
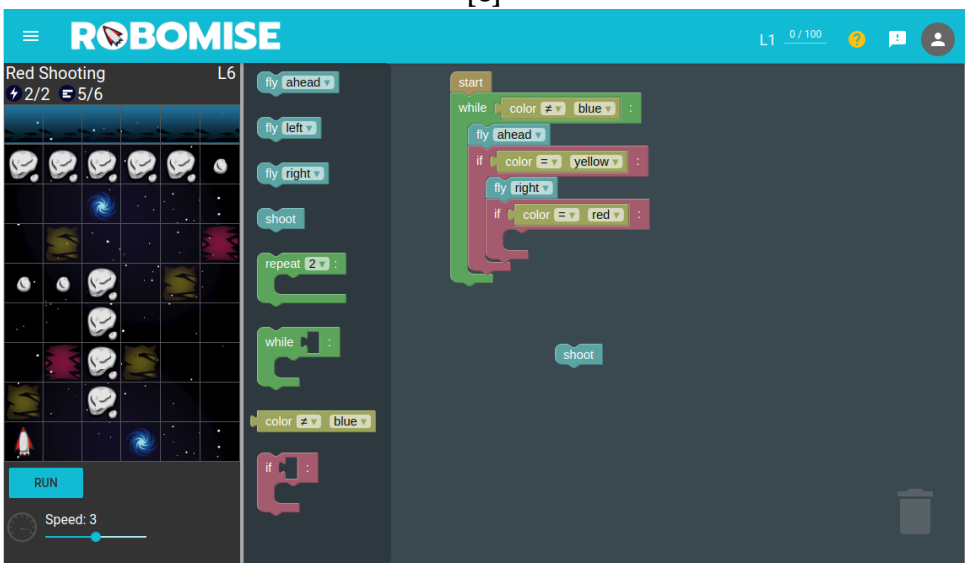


Figure 2.1: Introductory programming e-learning systems, part 1/2
(a) Hour of Code: Code with Anna and Elsa, (b) Robotanist

[c]



[d]

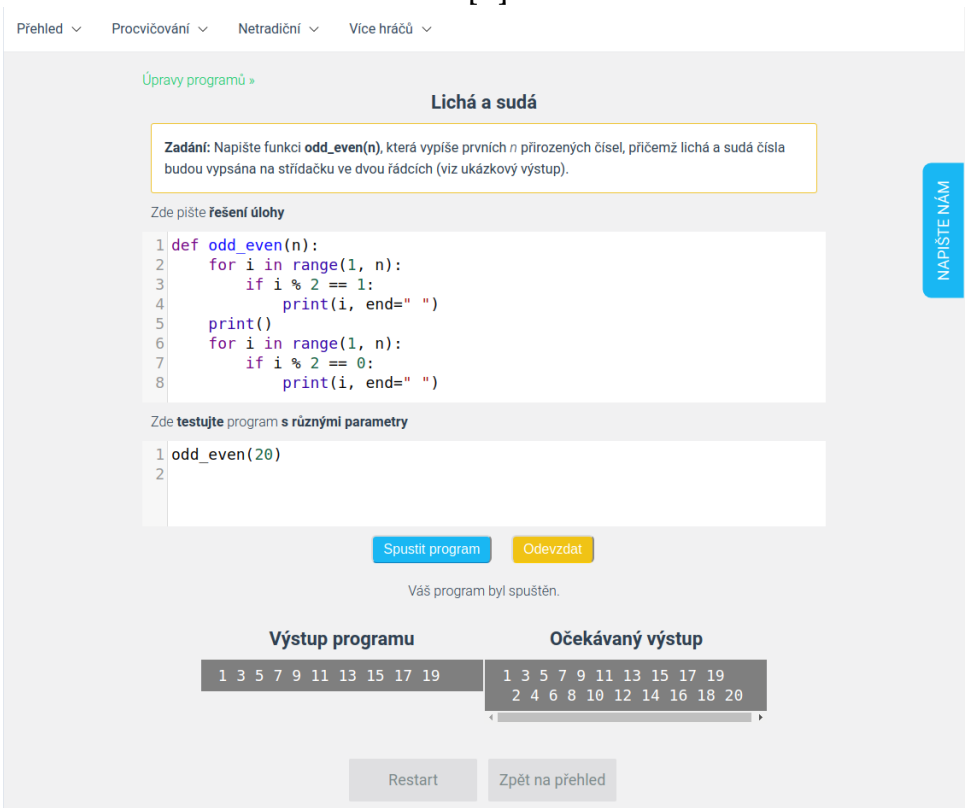


Figure 2.2: Introductory programming e-learning systems, part 2/2
(c) RoboMission, (d) Interactive Python

Table 2.1: Found measures and characteristics, part 1/3

no.	name	papers	relevant
TIME			
01	working time	[2], [3], [4], [5]	*
02	time between submits/compilations	[6]	*
03	time between compilation errors	[6]	
04	percentage of time in non-compiling state	[2], [3]	
05	compile error correction time	[3]	
06	block-editing-mode time ratio	[3]	
EDITS			
11	total keystrokes	[2]	
12	percentage of keystrokes in non-compiling state	[2]	
13	total lines of code	[2], [3], [7], [8]	*
14	code update pattern[a]	[9]	
15	time change of code update pattern	[9]	
SUBMITS, COMPILATIONS			
21	particular submits/compilations	[4]	*
22	total submits/compilations	[4], [10]	*
23	total successful/unsuccessful compilations	[4]	
24	percentage of successful/unsuccessful compilations	[4]	

a. Number of lines added/deleted/modified.

Table 2.2: Found measures and characteristics, part 2/3

no.	name	papers	relevant
CODE CONTENT			
31	total control flow statements	[2], [7]	*
32	total function definitions	[7]	
33	total variables	[7]	
ERRORS			
41	particular compilation errors	[4]	
42	most common compilation/run-time errors	[4]	*
43	total compilation errors	[4]	
44a	error quotient[a]	[6], [11], [12]	
44b	Watwin (error quotient improvement)[b]	[11], [12]	
45	error category	[5]	*
SCORE			
51	score/points/grade	[6], [7], [9], [11], [13], [14], [15]	*
CORRECTNESS			
61	correctness	[5], [11]	*
62	tests passed	[11]	
UNIQUENESS			
71	unique unit test outputs	[10]	
72	unique abstract syntax trees	[10]	*

a. The weighted sum of pairs of consecutive erroneous compilations with the same and different type of error.
b. The weighted sum of pairs of consecutive erroneous o erroneous-correct compilations with the same/different type of error, with same/different full error message, with an error on the same/different line combined with time between these compilations with respect to the mean time of this error type.

Table 2.3: Found measures and characteristics, part 3/3

no.	name	papers	relevant
TASK SIMILARITY			
81	bag of words of task description	[9], [14]	*
82	API calls similarity	[9], [14]	*
83	abstract syntax trees similarity	[9], [14]	*
84	concepts contained similarity	[16]	*
85	Levenshtein distance	[16]	*
86	n-grams overlap	[16]	*
87	input-output behavior similarity	[16]	*
88	program dependence graph structure	[16]	
89	code-state transition graph node similarity	[13]	*
MODE OF STATISTICS			
91	absolute stats for learner	all	*
92	relative stats for learner[a]	[4]	*
93	absolute stats for class	[4]	

a. Learner's class rank.

some categories belong to a single research question, others spread across a number of them. E. g. the time measures might be used to express both complexity of a task and a learner's task performance while the uniqueness category covers only task-uniqueness research question.

2.3 Dashboards

The main purpose of dashboards is to provide useful information in a well-arranged way. During the research of so-far-created introductory programming dashboards in literature, we found one dashboard of our type, i. e. designed for non-real-time, once-in-a-while use, and three dashboards which provide real-time information. All of these dashboards were developed for use in university introductory courses of Java programming.

Diana et al. [13] introduced a real-time teacher dashboard which computes and visualizes the predicted score of each student (see Figure 2.3); it is developed for student classes which solve three programming tasks. This dashboard displays time, the ratio of the number of students who started solving each task, the predicted score of all students and code snapshots of a particular student. The predicted score serves as information about students who finished their work, who are idle and (mainly) which students are struggling. These struggling students should receive some help from the teacher or from the well-performing colleagues.

Murphy et al. [4] presented a real-time dashboard for both teachers and students which is focused on compilations and executions of Java code (see Figure 2.4). Students can browse basic statistics of tasks (the total number of compilations, successful compilations, compilations errors and the time spent) computed for themselves and the class average, and an overview of their compilation and runtime errors. Teachers can see all this information computed for each student and the whole class.

Matsuzawa et al. created two dashboards. The first one [17] is a real-time dashboard of a student's code (see Figure 2.5) – students and teachers can replay the process of the student's code construction and monitor the working time, the number of lines, and the num-

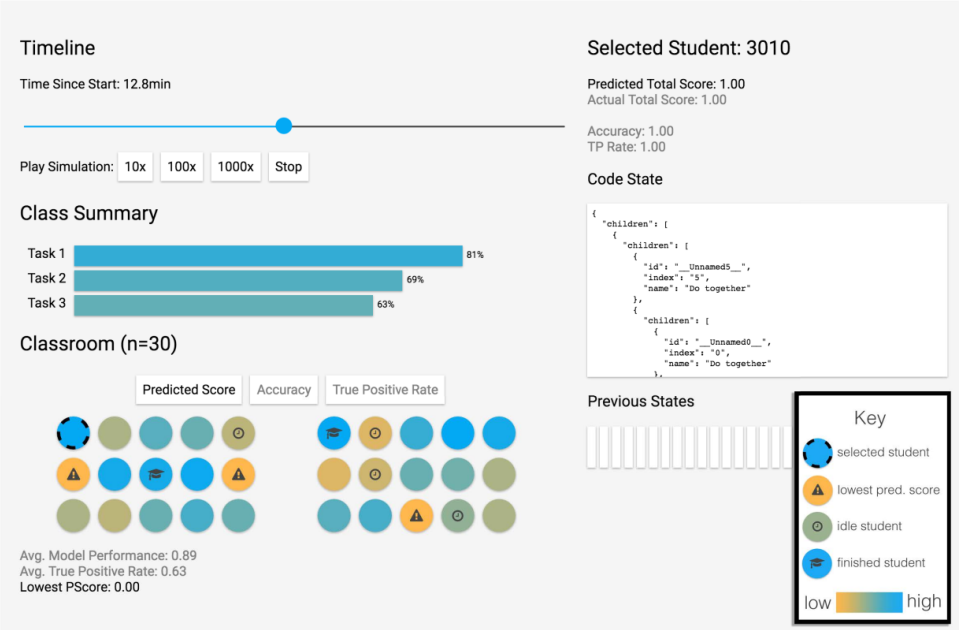


Figure 2.3: Real-time dashboard by Diana et al.

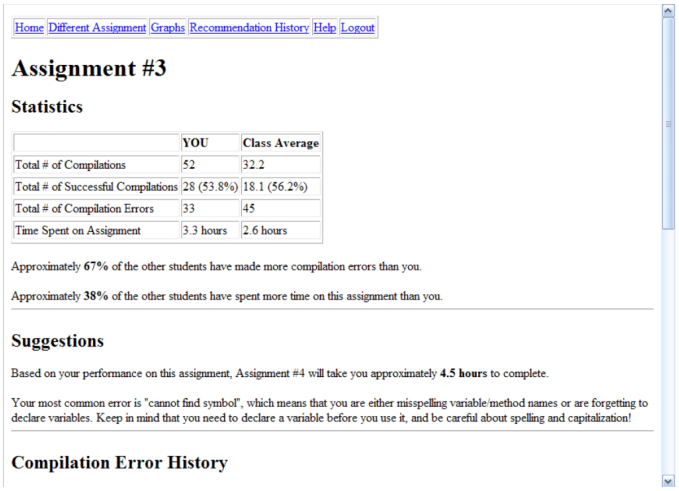


Figure 2.4: Real-time dashboard by Murphy et al.

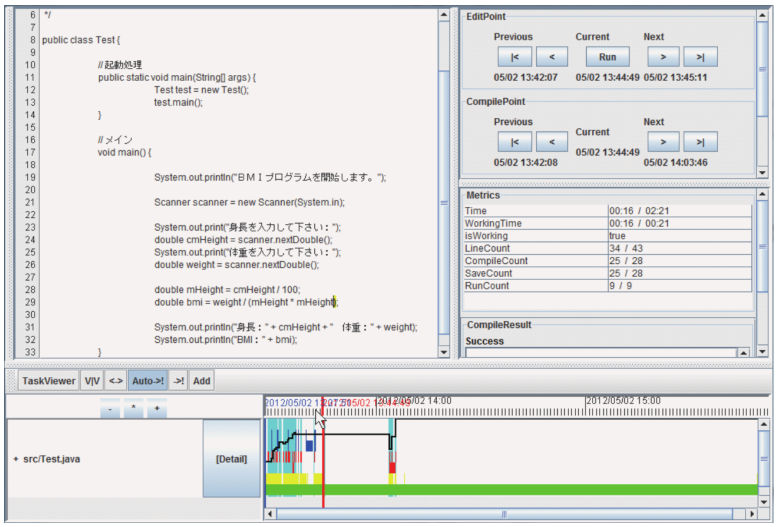


Figure 2.5: Real-time dashboard by Matsuzawa et al.

ber of compilations, saves and executions. Each operation with the code is displayed on the timeline panel. The second dashboard [18] is intended for teachers of a course with both textual and block environments, and the frequency of its use is estimated to once per a week or less (see Figure 2.6). It displays data from five programming measures – the compile error fixing time, the working time, the number of lines and the block editor usage[6]. The dashboard contains scatter plots of measures pairs, line charts for the values of the measures in each assignment and school year, distributions of measures values for students at each assignment, and the block-editing ratio for each student and assignment.

6. The fifth measure is uncertain; its description is not present in the text, and the snapshot of the dashboard is in Japanese.

Figure 2.6: Non-real-time dashboard by Matsuzawa et al.

3 Data and data processing

This chapter is concerned with the description of our e-learning system RoboMission (Section 3.1), its data we process (Section 3.2), the preprocessing of these data (Section 3.3) and the general description of the whole process of determining which measures are able to successfully answer the research questions mentioned in Chapter 1 (Section 3.4).

3.1 RoboMission

RoboMission[1] is an adaptive web e-learning system which has been developed by Tomáš Effenberger et al. at Faculty of Informatics, Masaryk University [19]. It teaches its users the basic principles of algorithmic thinking, such as loops and conditions, in a Python-like programming language with a visual drag-and-drop interface. A program consists of prepared blocks from Blockly[2] JavaScript library. A snapshot of RoboMission task solving interface can be seen in Figure 2.2[c].

The main task of a learner is to navigate a spaceship through a square-grid game board. In each step, the spaceship moves one row forward; a learner determines the kind of a move (forward, shoot and forward, diagonally to the left, diagonally to the right). Obstacles on the way comprise of asteroids (the spaceship crashes if present on the same square), smaller meteoroids (asteroids which can be shot down), wormholes (transport spaceship to other places), diamonds (all of them must be collected by the spaceship) and a limit on the length of the program.

Tasks are grouped into nine levels; each level consists of 3 phases based on the task difficulty. Learners practise new programming concepts or their combinations in each level. A recommendation of the next task provides adaptivity; a learner is guided through the system's levels and phases. A learner's performance on each answered task is evaluated by one of four labels based on the correctness and the solving time – incorrect, poor, good and excellent. Each of the last three

1. Website https://www.robomise.cz/ ; currently only in Czech.
2. Website https://developers.google.com/blockly/ .

labels implies an increase of skill in progress bars of the level and the phase; a learner is recommended a task from the next phase/level if the progress bar of the current phase/level is full. More information in [19].

3.2 Data

Our data consist of information about tasks, learners' task sessions, and code state logs. The term "task session" refers to an attempt (session) of a learner to solve a task. A learner may have 0, 1 or (rarely) more task sessions of one task.

Table 3.1 displays the basic data statistics; the distribution of task sessions among tasks and learners is described by Figures 3.1 and 3.2. The task sessions distribution among tasks is grossly unequal; the first three levels make up three-quarters of all task sessions. Learners typically try to solve only a small number of tasks – the median of the number of learners' task sessions is 12, this corresponds to the first three levels' prevalence in task sessions data.

Program code states are logged whenever a learner performs one of the two possible operations with code – edit or submit. An edit means a change of the program code; a submit stands for an execution of the code. If the code execution is correct, the task is considered solved, and the next task is recommended to the learner.

A program code is logged in its shortened form called *MiniRoboCode*. This shortened language uses a one-character abbreviation for each type of blocks:

- `"f"`, `"l"`, `"r"` and `"s"` for commands `fly forward`, `fly left`, `fly right`, and `fly forward and shoot`,
- `"R"`, `"I"`, `"/"` and `"W"` for control flow statements `repeat`, `if`, `else`, and `while`,
- `"b"`, `"k"`, `"y"`, `"g"`, `"r"` and (in)equality signs for blue/black/yellow/green/red square-color tests,
- `"x"`, numbers, and (in)equality signs for x-position tests,
- numbers for the number of iterations,

Table 3.1: Data statistics

Learners	3,503
Tasks	85
Task sessions	63,317
Program code logs	976,712

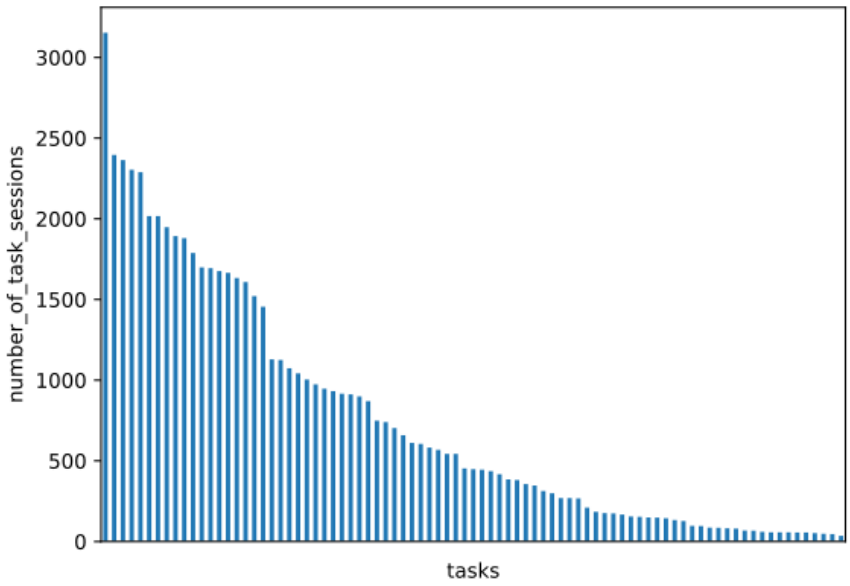


Figure 3.1: Number of task sessions for each task

- curly brackets for bodies of control flow statements.

E. g. the program `while square_color != blue: {if x_position > 3: {left} else: {repeat 4-times: {forward}}}` is transformed into the MiniRoboCode form `W!b{Ix>3{l}/R4{f}}`.

3.3 Data Preprocessing

The first thing needed for proper processing of data was a slight modification of all logged programs; MiniRoboCode uses symbol `"r"` for two blocks – `right` and `red`. This ambiguity causes problems in data

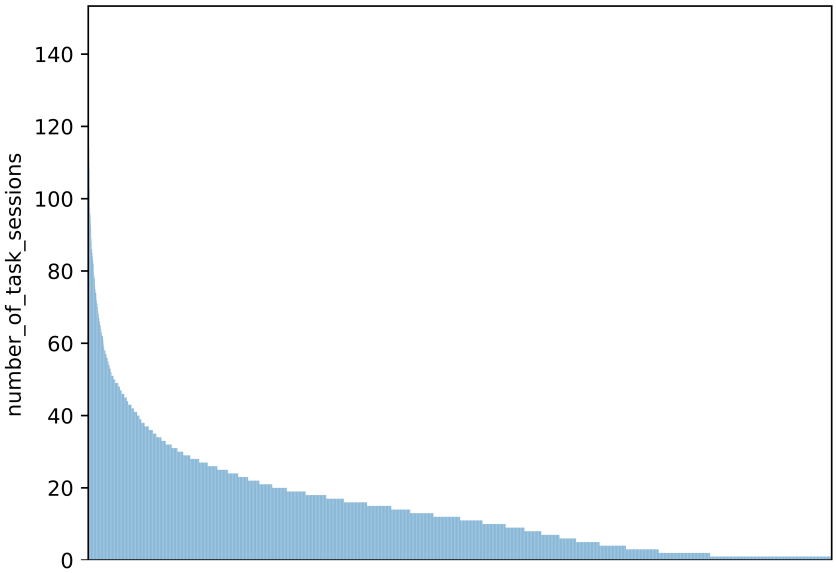


Figure 3.2: Number of task sessions for each learner

processing and therefore it is advisable to remove it. The correction is easy, `red` block is always followed by `"{"` sign and `right` block never occurs at this place, but this replacement is very time-consuming for the whole dataset. Our recommendation for RoboMission developers is to remove this ambiguity in log files and to change settings of MiniRoboCode-creating software for future logs.

During the further data processing, inconsistency in data was found. Some of the submitted programs had been evaluated incorrectly – programs with the correct solution were rejected while some incorrect ones were not. This state involved 1230 cases, thus circa 0.6% of all submits.

The investigation found two causes of these discrepancies. The first one is a recycling of a task identifier – one task was removed from the system, and then a new one with the same identifier was created, but the previously logged data of the original task still referred to the old identifier. This caused a 98.5 % of rejected correct programs. The second cause of discrepancies is more complicated than the first one. The code submitted by a learner is sent to the interpreter to be

evaluated. However, a graphical web interface needs a code evaluation block-by-block in order to highlight currently-processed blocks of the learner's code. This interpreter asynchrony may result in problems if a non-standard event happens in the communication between the learner's device and the server, e. g. if the learner leaves the web page before his or her code is completely processed. The asynchronous interpreter problem is believed to produce a majority of incorrect programs acceptance, but we found no proof of this statement.

As a result of found discrepancies, a synchronous Python interpreter was written for MiniRoboCode. We executed all the submits in data with this new interpreter and got new, correct information about the correctness of programs. Moreover, we recorded also trajectories of the spaceship through the game board which provided us information about the output behavior of submitted programs. All the program logs whose correctness according to the old and new interpreters does not agree were discarded.

3.4 Methodology

When trying to find answers to the research questions mentioned in Chapter 1, the first step is to find reasonable characteristics which could, at least hypothetically, provide the answer. The process of seeking these measures starts with examining measures found in the literature (Section 2.2). They are evaluated in the sense of their usability in RoboMission – e. g. RoboMission does not incorporate compilations, so the characteristics which count with compilations are not usable. The measures which are found generally-relevant are filtered by each research question (and slightly modified if needed). These sets of characteristics are then extended by measures created literature-independently in order to fill gaps among the measures and to cover RoboMission's specifics. Particular measures are described in chapters and sections corresponding to each research question.

All the measures are then computed on RoboMission data, and some of them are selected to be represented in the dashboard. An ideal measure carries important information, is easy to understand, is easy to be computed, acquires a sufficiently wide range of values, does not

suffer from ambiguity problems, its distribution is not very skewed, and it does not strongly correlate with other displayed variable.

In order to satisfy the last condition, the initial method of measures analysis is the correlation analysis. It often reveals groups of measures which perform similarly – then only one of the measures in a group is selected to the final dashboard since the others would bring only a little new information compared to the selected one. We analyzed correlations among the measures at three levels[3].

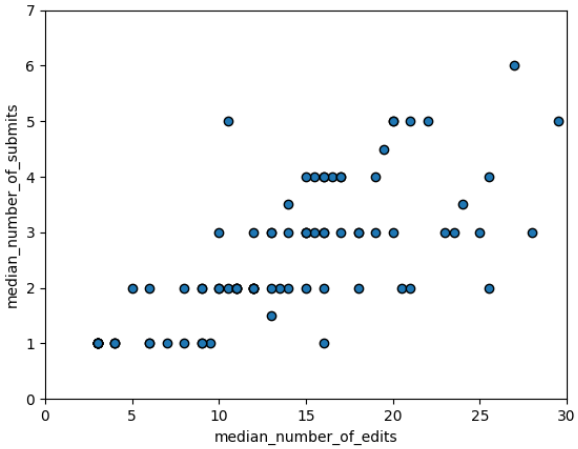
The first level is the correlation of values of two measures; we compute both measures for all tasks or learners, and the resulting two numerical sequences are compared by the correlation coefficients computation as seen in Figure 3.3. When computing a correlation of two variables (measures), there are two main correlation coefficients we could use – the Pearson's and the Spearman's ones. Both of them were computed and compared. They perform similarly in many cases, but they differ if the correlation between variables is not linear – e. g. measures incorporating time grow asymptotically faster than some other measures. In these cases, the Spearman's coefficient shows a significantly higher correlation than the Pearson's one.

The second level includes correlations of all the measures pairs and creates a correlation table of measures, as in Figure 3.4.

In order to objectively quantify the difference between Pearson's and Spearman's correlation coefficients, we constructed the third level of measures correlations. This third level computes the correlation of correlation tables; it flattens both the Pearson's and the Spearman's correlation tables to single lines and treats them as numerical sequences. Then it computes Pearson's correlation of these sequences. Here, only one type of correlation coefficient was chosen for simplicity.

However, there is one more dimension to decide how to measure the third level of correlations – since the correlation tables are symmetrical with 1s on the main diagonal, the redundant elements on the main diagonal and above it may shift the correlation coefficient to higher values. The significance of this phenomenon was not known. Therefore two variants of the third level computations were performed. The first one works with full correlation tables (Figure 3.4), the second one only with elements under the main diagonal (Figure 3.5).

3. Not to be confused with levels in RoboMission.



-> Pearson's correlation 0.73
-> Spearman's correlation 0.76

Figure 3.3: First level of measures relation

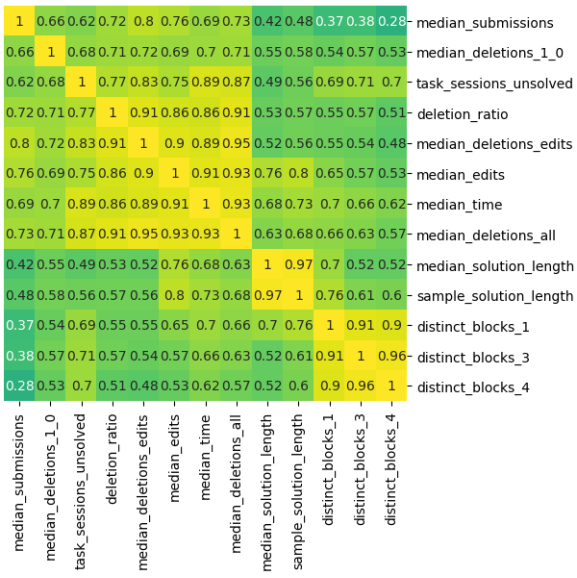


Figure 3.4: Measures correlation table; the full form

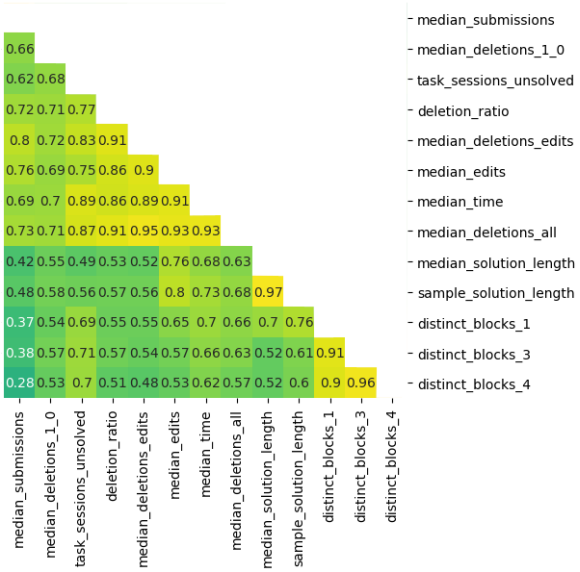


Figure 3.5: Measures correlation table; the lower-triangle form

Our analyses showed that the difference between the correlation of full and lower-triangle correlation tables might be significant in cases of some research questions. The full-table method was always producing higher values; therefore the lower-triangle-table method was used for the determination of the best first-level method (the Pearson's or the Spearman's one).

The choice of the first-level correlation coefficient plays a role at three research questions measures sets (according to the lower-triangle-table method); two of them contain a time variable. For this reason, the Spearman's correlation coefficient which covers even the non-linear correlation was chosen to represent measures' correlations in the first level. Results of the second and third levels analyses are presented in Appendix B.

After the correlation analysis, the other criteria for dashboard measures selection are applied with respect to the particular measures.

4 Task Characteristics

This chapter is concerned with research questions applied to tasks. Tasks information is important because a modification of tasks and their environment is the only way how to modify the entire e-learning system since a system developer has no direct control of learners.

4.1 Task Difficulty and Complexity

Our concept of a difference between the task difficulty and the task complexity follows on from the work of Sheard et al. [8]. While the task difficulty lies in total demands necessary to complete a task, the task complexity concerns only with the cognitive demands. The difficulty is thus always greater than the complexity. Non-cognitive aspects which determine the relationship between difficulty and complexity are the length of a task, the task description, and the solving environment.

A basic mathematical derivative task (e. g. $(x^3)'$) is very complex, but its non-cognitive demands are very low – just to read the input and to write symbols of the solution. Therefore the complexity and the difficulty of this task would be very similar. A children's maze task is not complex, but it has high demands on a learner's concentration and orientation so the difference between the complexity and the difficulty would be high.

The motivation for measuring difficulty and complexity of tasks is the fact that this information may have an impact on the categorization of tasks into phases, on the creation of new tasks, and modification of current tasks whose difficulty and complexity measures are extreme or in unusual relation.

We selected nine measures which are associated with the task difficulty or complexity and divided them into two groups by the fact whether they are in most cases influenced only by non-cognitive aspects of difficulty or not (complexity measures are not).

The ratio of unsuccessful task sessions is a complexity measure computed as the number of task sessions without a correct solution divided by the number of all logged task sessions. This measure is equal to $1 - \textit{success rate}$ but the success rate's higher values mean a less difficult task. Our other measures work reversely so the success rate would

be penalized in the measures dendrogram we use for the correlation table visualization.

The number of distinct blocks used in the sample solution is a proxy complexity measure for the number of concepts which are necessary for the successful completion of a task. Three variants of this measure were created because blocks `forward`, `left` and `right` belong all to the single basic concept "moving" and the semantics of block `shoot` is very similar to the previous three ones (compared to the others, control statement blocks). Variant *blocks 1* treats all blocks as distinct, *blocks 3* considers blocks `forward`, `left` and `right` one block type and *blocks 4* treats all four blocks as one type.

All the other measures are influenced only also by non-cognitive aspects of difficulty.

The median of solving times is the median of all times which various learners spent during the solving of a given task.

The median of the number of edits is the median of the number of program-code-changing operations the learners performed while solving a task.

The median of the number of submits is the median of the number of attempts the learners performed to execute and evaluate the program.

The median of the lengths of correct solutions and *the length of the sample solution* measures describe the expected length of correct solutions of a task. The first one is computed from program snapshots logs; the second one comes from the static data about a task's description.

Deletion measures are based on the median of the number of shortenings of the program code. There are several ways how to compute deletions; we selected three of them. The simplest way is to recognize whether a learner shortened the program code at least once, this leads to the binary variable called *deletions 1/0*. The second way is the most similar to the general description of deletion measures, i. e. to sum all edit operations which shortened the program code; we call it *deletions edits*. The third way takes account of the fact that a learner may shorten the program code by more than one block; it sums up all the blocks that were removed during the task session, this way is called *deletions all*. The last deletion measure (*deletions ratio*) uses deletions 1/0 measure and computes how many learners shortened their code at least ones, relatively to the number of all learners who started solving a given task.

In the case of edits and deletions, we should remark that although a learner wants to modify a program by a single action (a change of one block), the logging of this action may lead to the number of operations different from 1. Both reasons for this state come from the properties of RoboMission's programming environment.

The first reason is related to modifications inside the program code. If a learner wants to remove a block surrounded by other blocks, it is not possible to remove it with a single operation. Every time a learner manipulates with a block, he/she also manipulates with all the blocks which follow the selected one. A leaner has to (1) split the program code into two segments, (2) remove the given block and (3) join both segments into one again. A single action is logged as three edit operations – remove all blocks from the second segment, remove the given block[1], add all blocks from the second segment. The number of edits is thus 3; the number of deletions is $n + 1$, where n is the length of the second segment, in the case of counting all blocks or 2 in the case of counting deleting edit operations or 1 in the case of binary use of deletions.

This case is also related to the second reason that only the part of the program code which is connected with initial block `start`, is logged. If a learner deletes some blocks from the separated segment of code, we have no information about it. Therefore the values of the number of edits and deletions variants could be even 2, n, and 1 if the given block is removed from the second segment.

Results of the correlation analysis can be seen in Figure 4.1 – it is a Spearman's correlation table dendrogram. Both complexity measures (the ratio of unsuccessful task sessions and the blocks group) do not correlate with each other significantly better than with the other measures. Also, both complexity measures are located in very distant branches of the dendrogram. Based on these findings, we conclude that concepts of task difficulty and complexity are so similar in RoboMission data that we can treat them all as measures of a single task aspect and consider the research question No. 2 as answered together with the question No. 1.

1. This operation is not logged if the given block is in the second segment which is not connected with the `start` block.

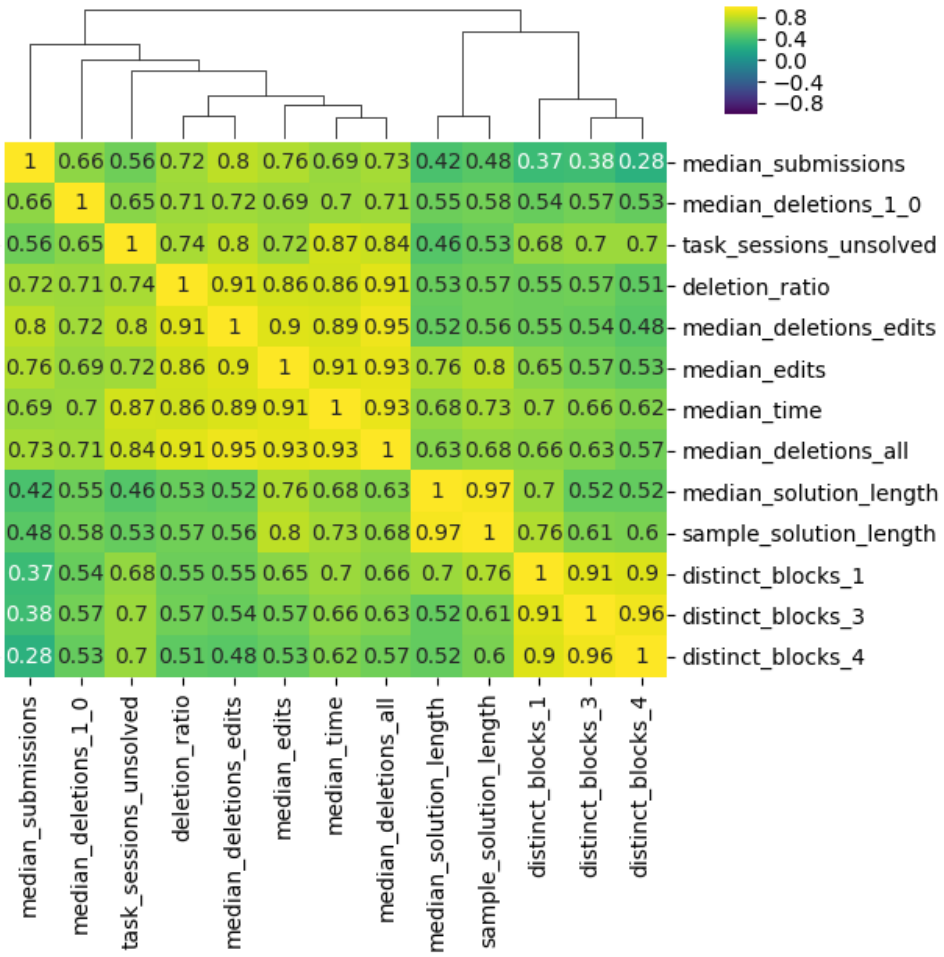


Figure 4.1: Difficulty and complexity measures – Spearman's correlation

There are three visible groups – the distinct blocks measures group, the solution lengths group and the other measures. They all correlate relatively well (in the case of the first two groups very well) within the group but significantly less outside.

Distinct blocks measures appear to be strongly dependent on the level to which a task belongs[2], according to our analyses. Since the number of practiced block types grows with levels, the information contained in distinct block count can be successfully estimated from the task's level information. The level does not need to be computed; therefore the block measures group is not used for the dashboard, and the level information is used instead.

The median of the lengths of correct solutions correlates with the sample solution length at 0.97. Moreover, the sample solution and the most-frequent correct solution are the same in approximately two-thirds of tasks. The sample solution length can be easily found out from the task's sample solution which has been already planned to be used in the dashboard, and that is why this group is not processed further.

The rest of measures is not derivable from static task characteristics. Deletion measures and the median of submits range only 6 or fewer values for the lower half of tasks (see Table 4.1). The number of edits is distorted by the problems mentioned above. Moreover, the ratio of unsuccessful task sessions and the median of solving times suffer from the attrition bias – the sample of people solving each task is not the same. The easiest tasks are solved by less skilled learners, and therefore they seem to be more difficult than they would be if solved by a representative sample of learners population. The opposite phenomenon arises from the most difficult tasks and more skilled learners. The median of solving times also suffers from the recording of the time a learner was doing some other things (chatting with friends, being away from the keyboard, etc.).

Based on the previous facts, we selected the success rate ($1 - \textit{the number of unsuccessful task sessions}$) for the dashboard. This measure is easily understandable, straightforwardly computable, its range

2. Correlation of the number of the level and all blocks measures is approximately 0.8; the number of the level and block measures would form their own branch in the dendrogram if the level information was added to the analyses.

Table 4.1: Difficulty and complexity measures statistics

measure	0.1 quantile	median	0.9 quantile
the ratio of successful task sessions	0.035	0.204	0.584
the number of distinct blocks – blocks 1	3	4	7
the number of distinct blocks – blocks 3	1	3	6
the number of distinct blocks – blocks 4	1	3	5
the median of solving times [sec]	27	119	292
the median of the number of edits	4	13.5	23
the median of the number of submits	1	2	4
the median of the lengths of correct solutions	3	6	10
the length of the sample solution	3	6	11
the median of deletions – all	0	5	15
the median of deletions – edits	0	2	5
the median of deletions – 1/0	0	1	1
the deletions ratio	0.324	0.760	0.900

is sufficiently wide, and it corresponds well with the common perception of the "difficulty" meaning.

4.2 Solution Uniqueness

Correct solutions uniqueness measures express how diverse correct solutions of a given task are. The expected state is that a task has more than one correct solution, one or a small number of these solutions is dominant, and the others are semantically similar to the dominant one(s).

Solution uniqueness information is usable for the analysis whether learners use concepts which are intended to practice. If a task has many dissimilar correct solutions, it may be proposed to rework the task.

The number of unique correct solutions expresses the uniqueness on the syntactical level of programs. All programs whose notation differ, are summed to this number although the meaning of these programs could be identical. E. g. the output behavior of the program `"while color != blue: {something}"` is at many tasks identical as of `"while color == black: {something}"` since the default square color is black and the final line color is always blue.

The number of visited-squares sequences relates to the output behavior of correct programs. When evaluating programs, sequences of squares visited by the spaceship are recorded. The comparison of these sequences is then used for purposes of detecting different programs from the view of their output behavior. E. g. programs `"forward; forward; forward"` and `"repeat 3-times: {forward}"` always lead to the same sequence of visited squares, and therefore their contribution to this measure is equal to 1.

The previous two measures are concerned only with the total number of different correct solutions. However, this does not say anything about the distribution of correct solutions. One of the possibilities how to express it is the *n-ary entropy* [20]:

$$H_n(X) = -\sum_{i=1}^{n} P(x_i) log_n(P(x_i)). \tag{4.1}$$

Variable n denotes the number of different classes (solutions) present in data. The probability of class i, $P(x_i)$, is not known. Therefore it is approximated by the observed frequency of solutions.

This method was used to obtain information about the distributions, in the form of *the entropy of the unique correct solutions distribution* and *the entropy of the unique visited-squares sequences distribution*.

The number of correct-solutions clusters measure uses a clustering of correct solutions. The idea behind this measure is to group programs by the code structure, not by the output behavior. The same output behavior can be reached in many ways, e. g. by a `for` loop or by various `while` loops which differ in stop conditions of the same meaning. The usage of different code elements often implies different idea behind the code so it might be valuable to distinguish these ideas.

Our clustering is based on abstract syntax trees (ASTs) and their tree edit distance (TED). Programs' ASTs are hierarchically clustered bottom-up into groups with the stopping parameter of TED-based cophenetic distance. A cophenetic distance is one of the ways how to measure distances between clusters – the distance of two clusters C_1 and C_2 is equal to the lowest distance of all C_1-C_2-elements pairs. Clusters are iteratively joined, ordered by the pairs with the smallest cophenetic distance. Once the cophenetic distances of all non-joined clusters are greater than 5, clustering is considered finished and no clusters are joined anymore. The stopping value 5 was selected empirically – it was the highest value which resulted into clusters that looked coherent, distance 5 is also the distance between programs `"while <condition>: {<something>}"` and `"repeat <repetitions>: {<something>}"` which differ only in one control element[3]. Distance value 5 was also used by Huang et al. in [10]. The number of clusters which were created by this procedure is taken as the measure's value.

Results of the correlation analysis are presented in Figure 4.2.

Measures of unique visited-squares sequences (its number and the entropy of its distribution) correlate strongly with each other. This is caused by the fact that the majority of tasks has only one seen correct visited-squares sequence (see Table 4.2). Therefore the entropy is zero,

3. E. g. correct programs of task No. 59 are grouped into two clusters. The first one contains programs `W!b{Iy{l}f}`, `Wk{fIy{l}}`, `W!b{fIy{l}}`, `W!b{fWy{l}}`, `W!b{sIy{l}}`, `Wk{sIy{l}}` and `W!b{Wy{l}f}`. The second cluster is comprised of programs `W!b{lR7{s}}` and `R2{lR7{s}}`.

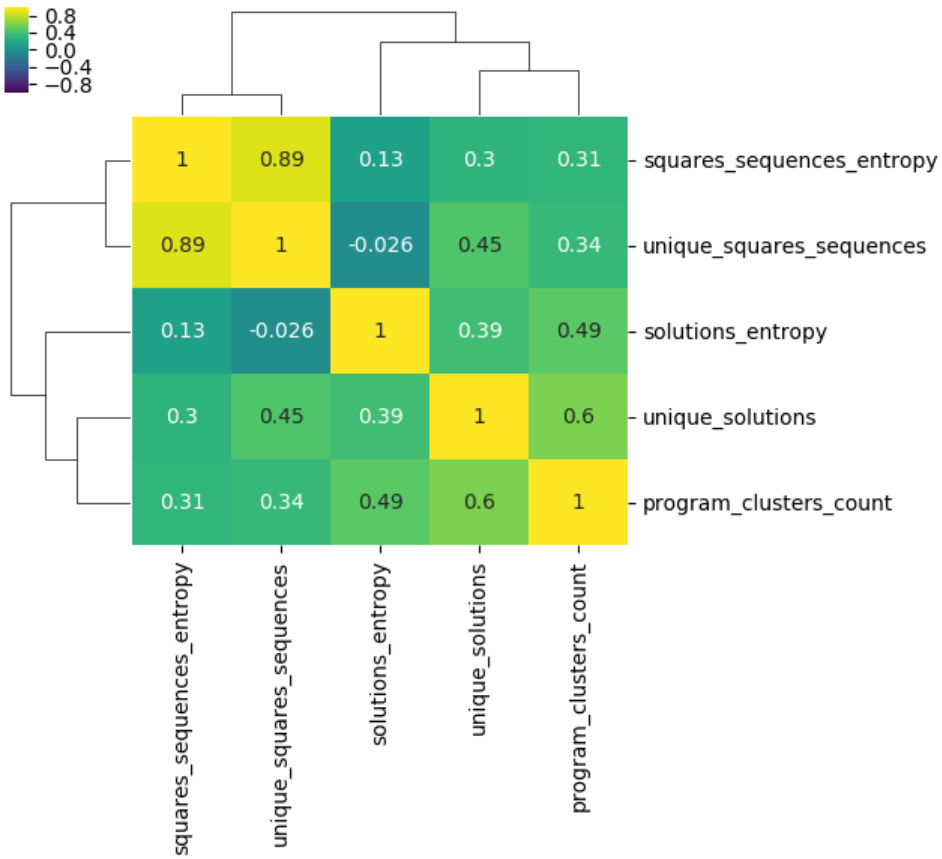


Figure 4.2: Solution uniqueness measures – Spearman's correlation

Table 4.2: Solution uniqueness measures statistics

measure	0.1 quantile	median	0.9 quantile
the number of unique correct solutions	2	8	32
the entropy of the unique correct solutions distribution	0.174	0.642	0.891
the number of unique visited squares sequences	1	1	5
the entropy of the unique visited squares sequences distribution	0.000	0.000	0.717
the number of correct solutions clusters	1	1	9

and thus correlation is high. The other measures do not form strongly correlated groups.

Tasks with the highest entropies usually belong to the last levels and have only lower tens of successful task sessions. An example is the task No. 39 from the ninth level, third phase; it has 19 successful task sessions distributed among ten correct solutions (in proportions 6, 3, twice 2, and six times 1) and three distinct visited-squares sequences (in proportions 11, 6, and 2). Particular solutions of this task can be seen in Appendix C.

Both entropy measures are not easily understandable. In the case of three measures – the number of unique visited-squares sequences, the entropy of unique visited-squares sequences distribution and the number of correct-solutions clusters measures – more than a half of tasks is described by the same triplet of their values[4]. Therefore, we select the number of correct solutions to the final dashboard. However, since the correlation of this measure with the others is not strong and since this measure is the summary one, some information present in data could be lost. To strengthen the information value of our dash-

4. Value 1 for the number of unique visited-squares sequences, 0 for the entropy of unique visited-squares sequences distribution and 1 for the number of correct-solutions clusters measures.

board, we add another feature – complete overview of all task's correct programs accompanied by their frequencies.

4.3 Task Similarity

Levels and phases in RoboMission are practised by many tasks. These tasks are supposed to be similar but not the same. Very similar pairs of tasks may be a signal of a mistake in tasks creation process – too similar tasks decrease a learner's motivation and learning effect. On the other side, a task extremely dissimilar to all the others may be either a suspicious one or a solitary task which could be accompanied by new tasks similar to this one.

We selected four source task features which measure the similarity between two tasks. However, it is much easier to measure distance (or dissimilarity) of tasks than their similarity; therefore all the selected features work reversely – the higher the value, the more dissimilar tasks.

The sample solutions distance features are based on sample programs – the programs are processed in various ways and then compared. In the first feature, *the tree edit distance of abstract syntax trees* (AST TED), a task's sample solution program is transformed into its abstract syntax tree, and tree edit distances of AST pairs are computed. The second feature, *the Euclidean distance of bag-of-blocks*, is based on counting program blocks' occurrences – the vector of these occurrences is called bag-of-blocks, the Euclidean distance of these vectors is used as a feature. *The Levenshtein distance of programs* is applied to plain MiniRoboCode programs to obtain the third feature; every addition, removal or change of a character of the first program to modify it to the other program is counted as +1 to the distance of tasks.

The game world distance feature, *the Euclidean distance of bag-of-game-world-entities*, describes the number of occurrences of all entities present on the game board – asteroids, meteoroids, diamonds, wormholes, colorful squares, and all squares – in the form of a vector. The dissimilarity of these vectors is then computed by the Euclidean distance. We are skeptical to the possibility of covering even the position of game world entities. The representation of positions, the selection of corresponding pairs of entities, problems with different game-board

sizes and with the distance between a present and a non-present entities[5], and the computation of distances of tasks include many ad hoc decisions which affect the result. There is also no clear evidence that positions should increase the information value of the feature. Therefore the information about entities' positions is not used.

These four features are used for the computation of a task dissimilarity matrix with $n \times n$ size where n is the number of tasks. The following two ways transform every row of the dissimilarity matrix to a single number which characterizes each task. The combinations of features and transformations (which also transform "1 to 1" dissimilarities to "1 to n" *similarities*) form our final set of task-similarity measures.

The distance to the closest task[6] is concerned with only the closest task according to the given feature, information about all other distances of the given task is discarded. This transformation defines the similarity of a task to all the others as a relationship between the given task and its *one* most similar neighbor. *The n-th percentile*[7] transformation computes the 2^{nd}, 5^{th} and 10^{th} percentile of all distances in the distance matrix. These values are then used as thresholds to the final measures where all the distances (within a given task row) which are lower than the threshold are counted. Numbers 2, 5 and 10 were chosen based on the number of tasks; expected numbers of similar tasks are thus 1.7, 4.25 and 8.5. These three measures try to define the term "similar task" by the threshold value and they count *all* the tasks within this threshold.

Results of correlation analysis are shown in Figure 4.3. There are at least three visible groups of measures – game world entities measures, blocks measures, and the others.

---

5. How to compute the distance between the asteroids if one task contains n asteroids while the other one does not contain any asteroid?

6. For the purposes of the correlation analysis, the *negative* distance to the closest task is used – the other transformation way uses the rule "the higher value the higher similarity", so this measure's values were inverted to the negative numbers in order not to be penalized by the correlation hierarchical-clustering algorithm. Outside the correlation analysis it is used in its original form, i. e. positive, "the higher value the higher *dis*similarity".

7. Where n equals to 2, 5 or 10.

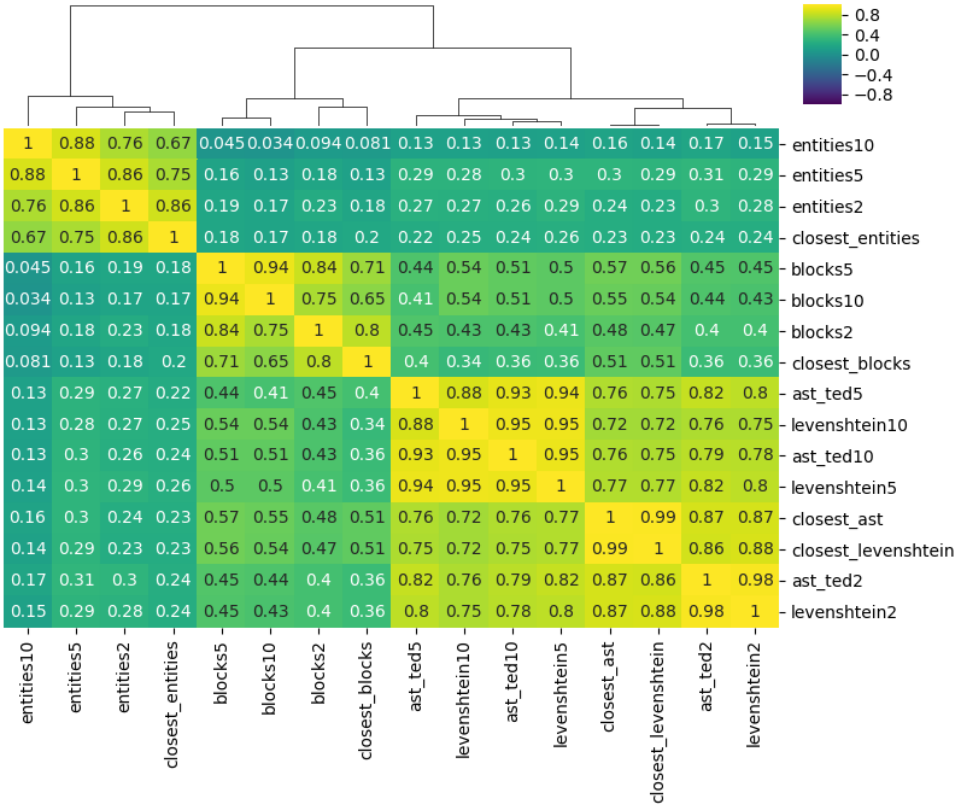


Figure 4.3: Task similarity measures – Spearman's correlation

The game world entities measures are concerned with the specification of a task. Thus, they are totally independent of the task solution. Moreover, this measure works only with occurrences of the entities, not with their position.

Here a question arises: Is, generally, the information about the game-world setting important? Our answer is: It could be but not for RoboMission developers and teachers. The same task can be created with two variants whose worlds are very dissimilar to each other – one task would be a normal one, the second would be the same as the first one with the addition of many redundant entities in the areas which are unreachable or not supposed to be visited. If we forced a maximal simplicity of the game world, a game-board similarity could be related with a difference of tasks' difficulties, but there is no such restriction. Sometimes, the presence of technically redundant entities is a developer's intention to improve the task aesthetically. For this reason, we do not classify game world entities measures as informative.

The bag-of-blocks measures do not take account of a code structure; they work only with vectors of blocks' occurrences. Since the semantic meaning of a program code does not lie only in used code elements but also in the way how they are combined, we deduce that other measures which reflect the structure of the code would be more informative.

The code-structure capturing measures form the third correlated group. AST TED measures work with full information about a program's structure while the Levenshtein distance measures flatten the syntax tree into the sequence of MiniRoboCode symbols. However, we can see that corresponding pairs of the Levenshtein distance measures and AST TED measures correlate very strongly with each other and also the basic statistics (see Table 4.3) of these two measures groups are very similar. This brings us to the finding that there is very little difference between the summary aspects of Levenshtein distance and AST TED measures groups. However, similar overall values do not mean that task pairs which are represented by these values are similar – distances to the closest tasks might be the same, but the particular name of the closest task might not (and similarly for the number of similar tasks). The AST TED measures use more detailed information about programs so its concept of the program similarity is more in-

Table 4.3: Task similarity measures statistics

measure	0.1 quantile	median	0.9 quantile
AST TED, 2^{nd} percentile	0	1	9
AST TED, 5^{th} percentile	0	2	15
AST TED, 10^{th} percentile	0	8	22
Euclidean bag-of-blocks, 2^{nd} percentile	0	2	7
Euclidean bag-of-blocks, 5^{th} percentile	0	6	12
Euclidean bag-of-blocks, 10^{th} percentile	1	11	18
Levenshtein distance, 2^{nd} percentile	0	1	9
Levenshtein distance, 5^{th} percentile	0	3	18
Levenshtein distance, 10^{th} percentile	0	6	22
Euclidean bag-of-entities, 2^{nd} percentile	0	1	5
Euclidean bag-of-entities, 5^{th} percentile	0	4	9
Euclidean bag-of-entities, 10^{th} percentile	0	9	17
AST TED, closest task	1	3	9
Euclidean bag-of-blocks, closest task	0.0	1.4	2.0
Levenshtein distance, closest task	1	3	9
Euclidean bag-of-entities, closest task	2.8	5.5	12.1

formative than the Levenshtein distance's one. Therefore we exclude Levenshtein distance measures from the further processing.

As for AST TED measures, we selected the 10^{th} percentile measure for the dashboard; it has significantly higher value of the median than the 2^{nd} and 5^{th} percentile measures whose median could be insufficient.

The previous AST TED measures expressed how much a task is similar to all the others; the last remaining measure tells us how much a task is dissimilar to its most similar task. This is a different aspect

of (dis)similarity, and therefore we include the dissimilarity of the closest task by the AST TED into the dashboard too.

4.4 Frequent Problems

A frequent problem point denotes a situation in which a significant number of learners struggle. Understanding these points helps developers or teachers to recognize the most problematic parts of each task. This information makes them possible to study whether learners have sufficient knowledge of the problematic concept or not. If not, teachers may discuss this topic with a class; developers may think about the ways how to force learners to practise the problematic concepts more.

The key point of the research question No. 5 is the definition of a frequent problem state. We work with program-states logs, so our definition is based on program states.

The recognition of a state in which a learner is in trouble is not easy. The perception of troubles is subjective, a troubled state of one learner might be a totally problem-free state for a different learner who has a different attitude to the solving of tasks. The recognition of these attitudes is difficult too because the majority of RoboMission learners tries to solve only a few tasks (see Figure 3.2). For these reasons, the problem state has to be defined universally for all learners. Our selected definition considers problems as a set of incorrect submits or unsuccessful task sessions.

The definition of the frequentness is very sensitive to the source of data – the distribution of incorrect submits is very different for a multiple choice question, a block programming task or a console programming task. Therefore the frequency threshold must be derived from data, the problem of setting this threshold is described below.

We selected two frequent-problems characteristics. They are not measures in the sense of their output – it is not a number but a list of frequent problems. We came out of the assumption that the core of the research question No. 5 is in the identification of the problem points, not in quantifying them.

The list of frequent incorrect submits characteristic uses all incorrect submits in the data. Every single incorrect submit is then considered a problem state.

The list of frequent leaving points characteristic uses only the last program-code states of unsuccessful task sessions. These programs (of both edit and submit types), "leaving points", are records of a state in which learners stop solving a task and leave its web page. The reasons for this behavior vary (difficulty – task too difficult for a learner, boredom – task too easy for a learner, time – end of the time allocated for RoboMission by a learner or a learner's supervisor, etc.), but a frequent point of leave might be a signal of a place where learners get stuck and consider the rest of a task too complicated.

The process of setting thresholds for frequent problems considers both absolute and relative frequency of programs. The main criterion of the frequentness is the relative ratio of a program's occurrences to the number of all task's problem states[8]. However, the size of data for each task differs a lot – the least-solved tasks have only around 70 incorrect submits and around 10 leaving points programs while the most-solved ones have over 10,000 incorrect submits and over 300 leaving points instances. In the case of tasks with the smallest amounts of data, only a relative threshold may lead to "frequent" problems with only several occurrences. Therefore a basic absolute condition to frequent problems is needed.

The data analysis showed that in the case of incorrect submits, even for the absurdly soft condition of relative threshold 0.06, the resulting percentage of frequent-problems logs in the all problems logs dataset[9] was below 10% for every value of the absolute threshold. The same result was in the case of leaving points with absolute threshold 8 and no relative threshold. Since this function is non increasing for increasing values of both thresholds, frequent problems would be paradoxically rare. The problem states are thus too varied in RoboMission, and the level of single-program code states is too fine-grained for the frequent-problems detection.

For this reason, some kind of program grouping is needed. We selected a grouping based on programs' output behavior equivalence, i. e. on the equivalence of visited squares sequences. The results of the

8. A program `foo` occurs in 9 problem states. The total number of all problem states is 90. The `foo` program has the relative frequency 0.1 and the absolute frequency 9.
9. Empty programs were excluded from these datasets.

Table 4.4: Thresholds for frequent-problem groups

	absolute threshold	relative threshold
incorrect submits	50	0.10
leaving points	10	0.10

grid testing of absolute and relative thresholds on grouped programs can be seen in Figures 4.4 and 4.5.

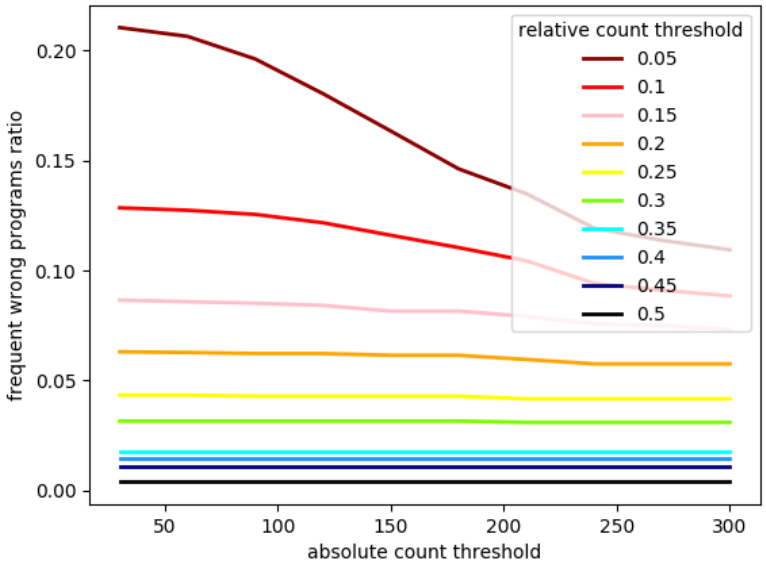
Based on this analysis, we set thresholds for frequent-problem groups as in Table 4.4. These thresholds imply the total relative frequency of frequent problems equal to 13% at incorrect submits, respectively 22% at leaving points. This values of thresholds were selected to satisfy the requirements that frequent problems should be frequent in both the tasks' scale and the total scale. However, there are even more values which satisfy given requirements, and these ones were selected based on our subjective judgment.

We created four measures in order to explore relations between our two characteristics with thresholds set as above: *the number of unique frequent incorrect submits*, *the number of unique frequent leaving points*, *the ratio of frequent incorrect submits* and *the ratio of frequent leaving points*.

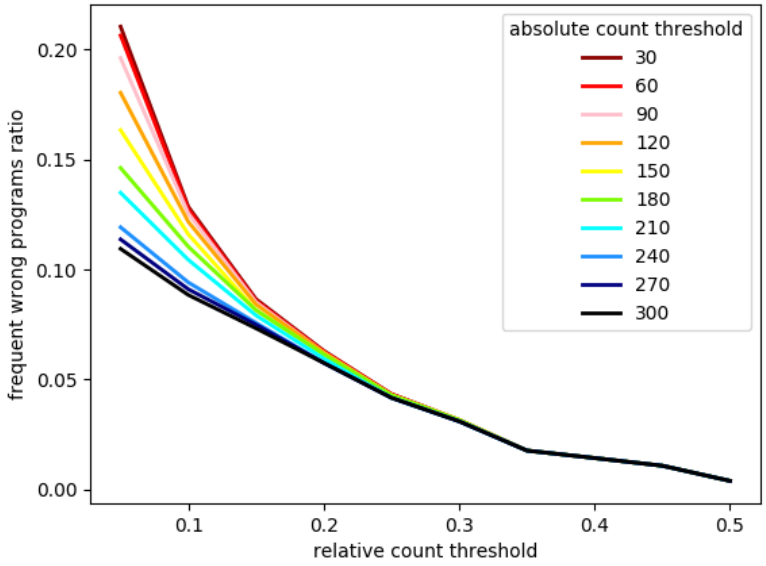
The first two measures describe the number of unique program codes whose occurrences satisfy given thresholds and thus are declared frequent. The last two measures express the total ratio of the number of frequent-problem-points logs and the number of all problem-points logs.

Results of analyses are presented in Figure 4.6 and Table 4.5.

Figure 4.6 shows that all measures are strongly correlated with their partners from the same characteristic (incorrect submits/leaving points) while they correlate significantly less with the other ones. Therefore we deduce that each of our characteristics could be informative in its own way. Alternatively, the size of leaving points data could be insufficient, so the noise in data could prevail the carried information and lower the correlation. Anyway, Table 4.5 shows that even with our soft thresholds the majority of tasks has no frequent problem point in both ways of its meaning. This finding resulted in adding both lists to the final dashboard but without restriction to only the frequent programs – all incorrect submits and all leaving

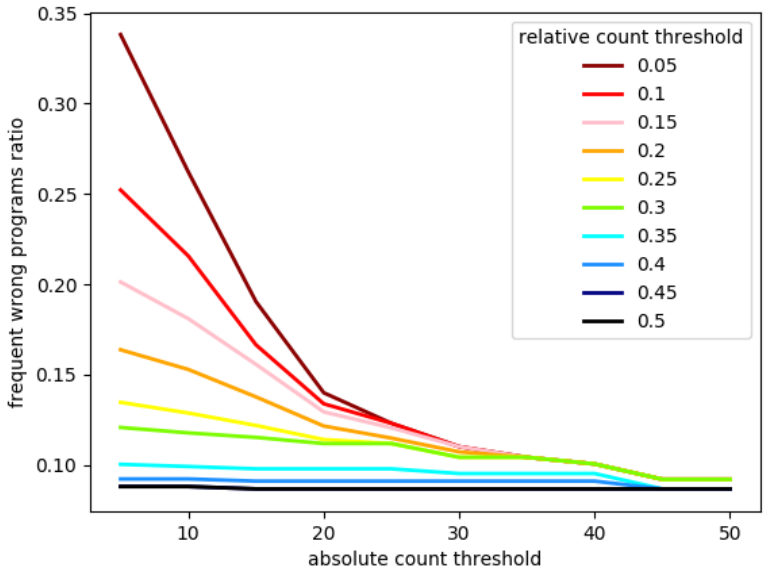


[a]

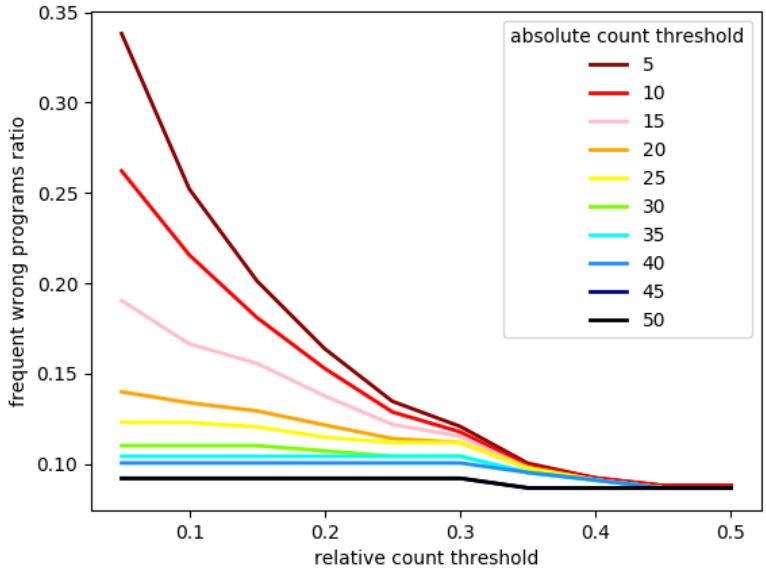


[b]

Figure 4.4: Incorrect submits frequency
(a) absolute frequency projection (b) relative frequency projection



[a]



[b]

Figure 4.5: Leaving points frequency
(a) absolute frequency projection (b) relative frequency projection

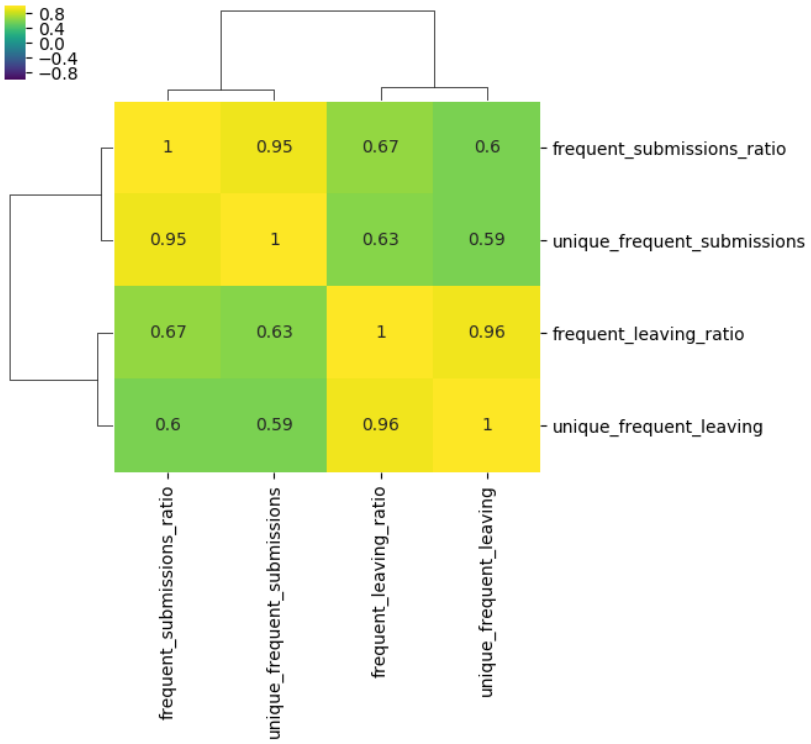


Figure 4.6: Frequent-problems measures – Spearman's correlation

Table 4.5: Frequent-problems measures statistics

measure	0.1 quantile	median	0.9 quantile
unique values			
the number of unique frequent incorrect submits	0	1	3
the number of unique frequent leaving points	0	0	2
the ratio of frequent incorrect submits	0.000	0.217	0.671
the ratio of frequent leaving points	0.000	0.000	0.521

points are listed in the dashboard, grouped by their visited squares sequences. Then it is up to a developer to make the decision which problem states are important. These squares-sequences groups are (for the purposes of the dashboard) represented by the most frequent program belonging to the given visited squares sequence.

5 Learner Characteristics

This chapter is concerned with research questions about learners. The importance of the knowledge of learners lies in the next-task recommendation process; it uses information about learners when determining which task is the most suitable for a learner at a given moment. This information consists of both learner's total performance and particular learner's performances on each task session. Reversely, this knowledge aggregated by tasks may produce new findings in the area of tasks.

5.1 Task Session Performance

A learner's performance on a task session is an essential variable used in many parts of RoboMission. It helps us to understand how much a learner is good at given concepts, how much difficult a task is, which tasks we should recommend to a learner, etc.

The most fundamental measure for the evaluation of a learner's performance on a task session is the correctness of a learner's answer. In RoboMission, this measure is binary – correct/incorrect. However, learners inside each of these two groups vary a lot – a learner with a straightforward way to the correct answer performs objectively better than a learner who struggles a lot, and the construction of the correct answer leads him or her to many dead ends. The similar principle holds in the case of unsuccessful learners who got stuck only a step by the correct solution and those who gave up in the beginning.

For these reasons, a more fine-grained measure is needed. We selected six task-session-performance measures which we analyze further.

The solving time (logarithm) measure is the time in seconds for which a learner was solving a task session. A pure time measure might be distorted by possible overvaluation if a learner was doing some other things while solving a task (chatting with friends, being away from the keyboard, etc.); therefore the *natural logarithm* is applied to the total time on a task session to get rid of these influences. The logarithm keeps the ranking of values and suppresses the effect of unnaturally long times. *The number of edits* and *the number of submits* measures

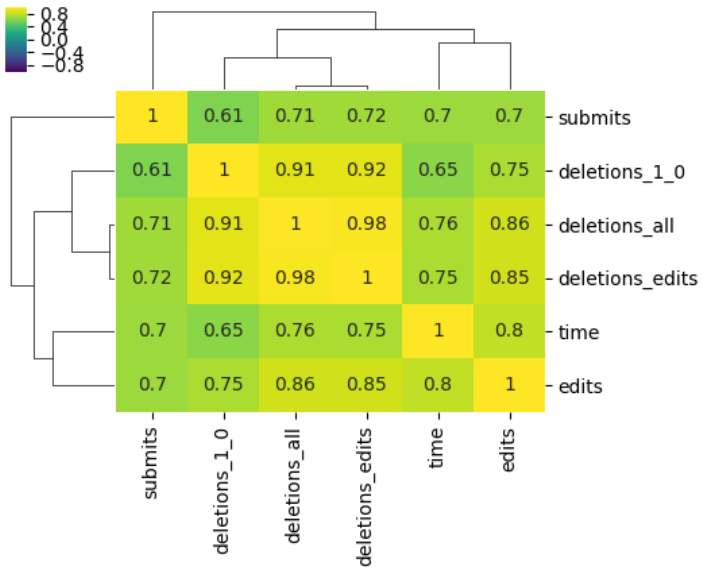


Figure 5.1: Learner's task session performance measures – Spearman's correlation

express the numbers of code-change and execution operations. We remind the information from Section 4.1 that not every single modification action results in one edit operation. It might be more operations if a learner modifies blocks in the middle of the code and it might be less operations if a learner constructs parts of the solution separately and joins them with the rest of the program later. In extreme situations, a learner may construct the whole solution aside and join it with the `start` block at the very end – this would lead to the solving the entire task with a single edit operation in logs. *The number of deletions* measures are computed from edit operations in the same way as in Section 4.1.

Results of the correlation analysis are shown in Figure 5.1. We can see that all measures correlate with each other at least 0.6.

Deletion measures correlate strongly with each other. The distribution of all of them is very skewed (see Table 5.1); more than a half of learners receives one of only two values (0 or 1). The number of edits may be distorted by problems with logging. However, these distor-

Table 5.1: Task session performance measures statistics

measure	0.1 quantile	median	0.9 quantile
the solving time (logarithm)	2.7	4.1	5.8
the number of edits	3	8	29
the number of submits	1	2	8
the number of deletions – all	0	1	18
the number of deletions – edits	0	1	7
the number of deletions – 1/0	0	1	1

tions seem not to be so significant; otherwise, this measure would not correlate well with the other measures. The range of the solving time is wider than the range of the number of edits and much higher than the range of the number of submits. We selected the logarithm of the solving time measure to the final dashboard.

5.2 Overall Performance

The research question No. 7 is concerned with a learner's performance in the whole system. This knowledge is valuable for the learner model used in the system – information about the learner's total progress determines the recommendation of the next task. Currently, measuring a learner's skill is performed as in Section 3.1 – it lies in aggregating data from a learner's task-session performances. However, a single-number metric for a quick overview is lacking.

The total number of solved tasks measure is a simple way how to measure a learner's progress – the further in the system a learner goes, the higher total number of solved tasks he/she has. On the other hand, the current recommendation system forces worse learners to practise more tasks of each level and phase than the better ones.

The total number of points gained refers to credits or experience points used in previous versions of RoboMission. These points were com-

Table 5.2: Overall performance measures statistics

measure	0.1 quantile	median	0.9 quantile
the total number of solved tasks	2	12	29
the total number of points gained	2	21	86
the total number of unique blocks used in correct solutions	3	5	8
the total solving time (logarithm)	6.5	47.8	125.7

puted as the sum of tasks solved in each level, multiplied by the corresponding number of level (1 to 9). This measure has similar properties like the previous one, but the weighting of levels should give an advantage to those learners who solve more difficult tasks.

The total number of unique blocks used in correct solutions is a proxy measure for the total number of concepts learned by a learner; the fact that a learner successfully used some block is considered as a proof of his/her knowledge.

The total solving time (logarithms) is a sum of natural logarithms of solving times of all the learner's task sessions. The reason for the logarithm is the same as in Section 5.1, but this time the influence of a single distortion is also reduced by summing all the learner's solving times.

Results of the correlation analysis are displayed in Figure 5.2. All measures correlate with each other at more than 0.8, so the other criteria selected the measure for the dashboard. The used-blocks measure ranges only around ten values (Table 5.2), the total solving time and the total number of solved tasks do not consider the difficulty of tasks. Therefore we selected the total number of points gained for the dashboard.

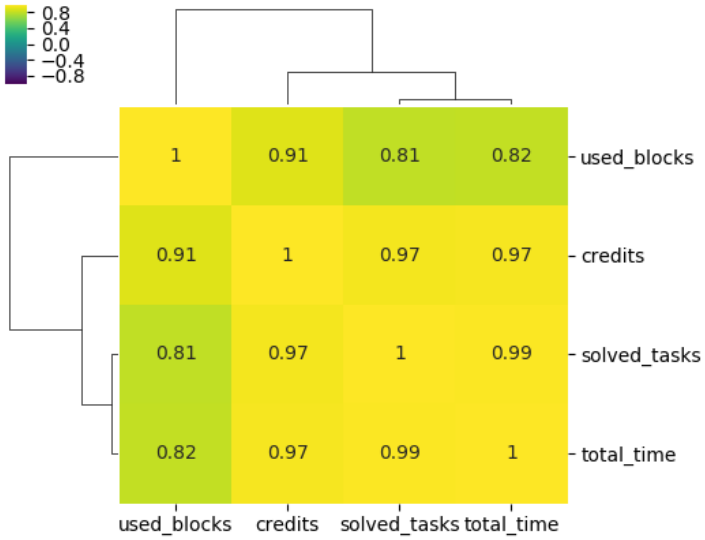


Figure 5.2: Learner's overall performance measures – Spearman's correlation

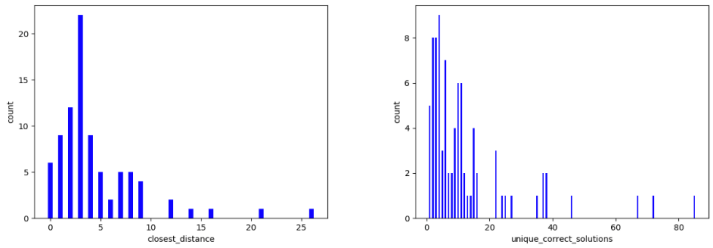
6 Dashboard

As mentioned in Chapter 1, the dashboard of measures which can successfully answer our research questions is intended to be one of the outputs of this thesis. Its purpose is to provide an overview of tasks' and learners' characteristics and to serve as a base for future analyses performed by RoboMission developers and teachers.

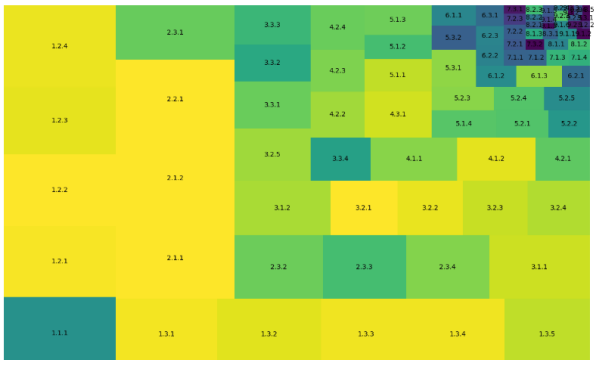
Although this dashboard is intended primarily for RoboMission developers, teachers who use the system in their classes are the second target group. However, the teacher-class mode has not been implemented in RoboMission yet. Teachers could appreciate more detailed information about students in their classes – a comparison of a student with the rest of the class, a comparison of the class with other classes, etc. Of course, a restraint on the visibility of other users would be necessary.

Our dashboard consists of six web pages. They describe our findings in the field of tasks, correct programs, incorrect submits, leaving points, learners' task sessions, and learners' total characteristics. Snapshots of the dashboard can be seen in Figure 6.1, the dashboard computed for data used in this thesis is present in Appendix D, Appendix A contains the program for its generation.

Each web page contains plots of distributions of its main variables and a table which constitutes the core of the page. An additional plot is presented at the tasks page; it is a tree-map which displays the number of task sessions and the success rate of each task. Tables consist of the index column or columns, variable columns, and additional columns. Index columns are identifiers of each row. The variable columns contain data we want to present; cells of these columns are colored by their values – the range of values between the 0.05^{th} and the 0.95^{th} quantile of a given variable is mapped to the *viridis* colormap. This colormap was selected because it is perceptually uniform – not only hue but also the lightness of its every color is unique. The brighter shades were selected for those values of variables which are better; however, in cases of some variables, there is no evidence which value is better. Values outside this range are colored as their value was equal to the 0.05^{th} /0.95^{th} quantile, i. e. by colors on the edge of the colormap's spectrum. This cut was performed to get rid of extreme values which could shift



Tasks by the Number of Task Sessions and the Success Rate



Tasks by number of task sessions (area) and success rate (color, the brighter the better)

Table

Filter columns by >/</= sign and the given value; e. g. '>35' or '=zig-zag'

Task	Name	Section	Sample Solution	Total Successful Task Session	Success Rate
AST TED 10	Closest Distance	Closest Task	Unique Correct Solutions		

Sort values by a click on the header

Task	Name	Section	Sample Solution	Total Successful Task Sessions	Success Rate	No. of close tasks, AST TED, 10th percentile	Closest Distance	Closest Task	Unique Correct Solutions
01	diamonds-in-meteoroid-cloud	3.1.2	R4{sr}f	1222	0.891	22.0	2.0	3.3.1	4
02	turning-left-and-right	1.2.4	lrf	2165	0.957	23.0	1.0	1.2.2	1
03	zig-zag	4.1.1	W!b{rl}	884	0.857	12.0	2.0	4.1.2	4
04	triple-slalom	9.3.1	R3{W!d{rl}f}f	13	0.31	2.0	2.0	9.1.1	6
05	diamonds-with-signals	6.1.2	W!b{fly{rl}}	197	0.631	14.0	1.0	6.1.1	16
06	tunnel	2.3.1	srlslrsff	1525	0.82	16.0	4.0	2.3.4	24
07	meteoroids-and-wormholes	7.1.3	W!b{Wx<5{r}sf}	83	0.748	8.0	4.0	8.1.2	22
08	shooting	2.1.2	rfssf	2171	0.977	22.0	2.0	2.1.1	11
09	diamond-lines	7.2.2	W!b{Ix=3{f}r}	76	0.5	15.0	3.0	7.1.1	7
10	shot	2.1.1	fsf	2293	0.99	22.0	1.0	1.1.1	6
11	ladder	3.1.1	R4{fs}	1517	0.927	22.0	3.0	3.1.2	11

Figure 6.1: Snapshots of the tasks web page of the dashboard

the color spectrum and leave parts of the spectrum underused. The additional columns serve as a source of additional information which helps the user of the dashboard to better understand and interpret the results.

The generation of dashboard's web pages is performed by Python library *yattag*, resulting pages use JavaScript library *sorttable*. The computation of data lasts about tens of minutes if we have solved the problem with the `"r"` symbol ambiguity (see Section 3.3) and pre-computed the correctness and visited-squares sequences for all submits in data. If not, the computation time extends to several hours. Therefore, we attach the pre-computed dataset in Appendix E.

As we have introduced the dashboard, it is time to demonstrate its usability on examples of possible analyses.

The success rate in *tasks* dashboard page shows us that tasks in phases 5.3, 7.1 and 9.1 have very various difficulty[1]. We might start thinking how to rework the tasks to be of similar difficulty, why the task 5.3.2 (No. 32) has a limit on the number of blocks while the task 5.3.1 (No. 47) does not, or whether we should change the mapping of tasks into phases.

The task No. 71 has an unusually high *number of unique solutions*. The further investigation shows that this task does not have number-of-blocks limit and therefore learners can create many corrects programs which combine single commands and loops in various ways. Tasks No. 76 and 18 have a large *number of unique solutions* too – values of their block limit are higher than necessary, and we might think whether to decrease them or not.

The task No. 60 has a relatively big value of its *closest distance*, and its sample solution is not very long. If we look closer, we can see that this task is the only task in the system which contains one `while` block and two `if` blocks in its sample solution. Creating more tasks with this combination of blocks could be beneficial.

The *closest task* to each of the tasks usually belongs to the same level as the source task[2]. Tasks from the level 5 have their closest tasks five times in the level No. 5, three times in 3, twice in No. 9 and once in

1. Phase 5.3: from 0.51 to 0.864; phase 7.1: from 0.518 to 0.748; phase 9.1: from 0.198 to 0.639.
2. However, there might be exceptions caused by the similarity of the first two levels and the recapitulation nature of levels No. 5 and 9.

the level No. 4. The level No. 5 is a recapitulation level for levels No. 3 and 4, so it is suspicious that tasks from level No. 4 are not the closest ones more often. If we look at sample solutions of level No. 5, we can see that all the tasks contain a `repeat` block in its sample solution, but only once they contain a `while` block. Level No. 5 practices both these blocks so we could recommend creating more tasks with `while` block in the level No. 5 to practice both concepts more equally.

The *incorrect submits* page shows us that the far most frequent wrong submission of the task No. 3 is `Wb{lr}`; the correct program would be `W!b{lr}`. We can deduce that learners do not understand the concept of `while` block condition sufficiently and then we can start thinking about how to explain this concept better to the learners.

The most frequent *leaving point* of the task No. 79 can be interpreted that learners did not know how to solve the task with only one shot and they left the task. The possible reason for this phenomenon is that the correct solution involves a left-diagonal move between stones located ahead and on the left. Learners may not know that this move is possible – some demonstration of this possibility could help them.

The information from the dashboard's web pages which are concerned with learners are intended to be used mainly in learner models; analyses of our learner measures in models would be model-sensitive and more complicated than the analyses presented above. Therefore we do not present particular analyses of these measures.

Our dashboard is very simple – its modification for other introductory programming e-learning system would be easy. The more difficult part of the different-system adaptation would be the analysis of informative measures which we have performed in Chapters 4 and 5 and the computation of these measures concerning the systems' specifics.

7 Conclusion

This thesis explores ways how to answer questions about measuring task and learner features in the field of introductory programming e-learning systems. These questions relate to the task difficulty, task complexity, solution uniqueness, task similarity, frequent learner problems, and the learner performance on a task and in the whole system. Particular analyses were conducted on data from the e-learning system RoboMission.

We performed research of introductory programming measures used in the literature and found 41 measures which were assessed from the point of their usability in RoboMission. Selected measures from the literature were divided by research questions to groups and extended by extra measures designed by us.

Each of the total 48 measures undertook the analysis of its correlation with other measures, its distribution, and its overall information value. The most beneficial measures were selected to be presented in the RoboMission dashboard.

We created a dashboard intended for developers which presents the results of our research on RoboMission measures. It contains key characteristics of tasks and learners represented in both graphical and table way. The dashboard can be modified in order to be used by teachers who use RoboMission in their classes.

During the research, we discovered interesting findings in RoboMission data, e. g. the ambiguity of the MiniRoboCode notation or problems with the evaluation of the code by an asynchronous interpreter. We also created various tools which may help RoboMission developers in future, e. g. a synchronous interpreter of MiniRoboCode or MiniRoboCode abstract syntax trees builder and comparator. These findings and tools were provided to the RoboMission authors in order to improve the behavior of the whole system.

Our dashboard and the entire analytic process are extensible for the programming concepts which are planned to be implemented in RoboMission in the future. They are also transferable to other introductory programming e-learning systems. However, it must be slightly modified concerning the system's specifics – the type of used program-

ming language, system's programming and output environments or the structure of logged data.

A Program Code

This appendix contains the Python code which was used for analyses and the creation of our dashboard – it consists of our code, slightly modified *betterast* package and JavaScript library *sorttable*.

B Correlations

This appendix contains visualizations of the second and third levels of correlations inspected in this thesis, as mentioned in Section 3.4.

C Solutions of the Task No. 39

This appendix contains solutions of the task No. 39, as mentioned in Section 4.2.

D Dashboard

This appendix contains the dashboard computed for the data we analyzed.

E Pre-computed Data

This appendix contains the pre-computed form of analyzed data – its files include the version of MiniRoboCode without ambiguity problems, the information about program correctness evaluated by the synchronous interpreter, and the information about visited-squares sequences for each submit in the data.

Bibliography

1. *NAE Grand Challenges for Engineering: Advance Personalized Learning* [online]. Washington, DC: US National Academy of Engineering [visited on 2018-12-01]. Available from: `http://www.engineeringchallenges.org/challenges/learning.aspx`.
2. IHANTOLA, Petri; SORVA, Juha; VIHAVAINEN, Arto. Automatically Detectable Indicators of Programming Assignment Difficulty. In: RUHTERFOORD, Becky et al. (ed.). *SIGITE '14 Proceedings of the 15th Annual Conference on Information technology education*. New York: ACM, 2014, pp. 33–38.
3. MATSUZAWA, Yoshiaki; TANAKA, Yoshiki; SAKAI, Sanshiro. Measuring an Impact of Block-Based Language in Introductory Programming. In: BRINDA, Torsten et al. (ed.). *Stakeholders and Information Technology in Education: IFIP Advances in Information and Communication Technology*. Cham: Springer, 2016, pp. 16–25.
4. MURPHY, Christian et al. Retina: Helping Students and Instructors Based on Observed Programming Activities. In: FITZGERALD, Sue et al. (ed.). *SIGCSE '09 The 40th ACM Technical Symposium on Computer Science Education*. New York: ACM, 2009, pp. 178–182.
5. PELÁNEK, Radek. Exploring the Utility of Response Times and Wrong Answers for Adaptive Learning. In: FITZGERALD, Sue et al. (ed.). *L@S '18 Proceedings of the Fifth Annual ACM Conference on Learning at Scale*. New York: ACM, 2018.
6. JADUD, Matthew C. Methods and Tools for Exploring Novice Compilation Behaviour. In: ANDERSON Richard; Fincher, Sally A.; GUZDIAL, Mark (eds.). *ICER '06 Proceedings of the second international workshop on Computing education research*. New York: ACM, 2006, pp. 73–84.
7. ALVAREZ, Andres; SCOTT, Terry A. Using Student Surveys in Determining the Difficulty of Programming Assignments: The Next Decade for Digital Libraries. *Journal of Computing Sciences in Colleges*. December 2010, vol. 26, pp. 157–163.

8. SHEARD, Judy et al. How Difficult are Exams?: A Framework for Assessing the Complexity of Introductory Programming Exams. In: CARBONE, Angela; WHALLEY, Jacqueline (eds.). *ACE '13 Proceedings of the Fifteenth Australasian Computing Education Conference*. Darlinghurst: Australian Computer Society, 2013, pp. 145–154.
9. BLIKSTEIN, Paulo et al. Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. *Journal of the Learning Sciences*. November 2014, vol. 23, pp. 561–599.
10. HUANG, Jonathan. Syntactic and Functional Variability of a Million Code Submissions in a Machine Learning MOOC. In: WALKER, Erin; LOOI, Chee-Kit (eds.). *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013*. Aachen: CEUR-WS.org, 2013, pp. 25–32.
11. AHADI, Alireza et al. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. In: DORN, Brian; JUDY, Sheard; QUINTIN, Cutts (eds.). *ICER '15 Proceedings of the eleventh annual International Conference on International Computing Education Research*. New York: ACM, 2015, pp. 121–130.
12. WATSON, Christopher; LI, Frederick W. B.; GODWIN, Jamie L. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In: *ICALT '13 Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies*. Washington, DC: IEEE, 2013, pp. 319–323.
13. DIANA, Nicholas et al. An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments. In: WISE, Alyssa et al. (ed.). *LAK '17 Proceedings of the Seventh International Learning Analytics Knowledge Conference*. New York: ACM, 2017, pp. 272–279.
14. PIECH, Chris et al. Modeling How Students Learn to Program. In: KING, Laurie S. et al. (ed.). *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education*. New York: ACM, 2012, pp. 153–160.
15. PODGORELEC, Vili; KUHAR, Saša. Taking Advantage of Education Data: Advanced Data Analysis and Reporting in Virtual Learning Environments. *Electronics Electrical Engineering*. October 2011, vol. 114, pp. 111–116.

16. WALENSTEIN, Andrew et al. Similarity in Programs. In: KOSCHKE, Rainer; MERLO, Ettore; WALENSTEIN, Andrew (eds.). *Dagstuhl Seminar Proceedings: Duplication, Redundancy, and Similarity in Software*. Wadern: IBFI, 2007. No. 06301.

17. MATSUZAWA, Yoshiaki; OKADA, Ken; SAKAI, Sanshiro. Programming Process Visualizer: A Proposal of the Tool for Students to Observe Their Programming Process. In: CARTER, Janet (ed.). *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. New York: ACM, 2013, pp. 46–51.

18. MATSUZAWA, Yoshiaki et al. A Demonstration of Evidence-Based Action Research Using Information Dashboard in Introductory Programming Education. In: TATNALL, Arthur; WEBB, Mary (eds.). *Tomorrow's Learning: Involving Everyone: Learning with and about Technologies and Computing*. Cham: Springer, 2017, pp. 619–629.

19. EFFENBERGER, Tomáš. *Adaptive System for Learning Programming*. 2018. Master's thesis. Masaryk University.

20. SNEED, Joseph D. Entropy, Information, and Decision. *Synthese*. 1967, vol. 17, no. 1, pp. 392–407.