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 pelanek, Ph.D.

Acknowledgements

These are the acknowledgements for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can
span multiple paragraphs.

Keywords

introductory programming, adaptive learning, dashboard, RoboMission

Contents

1 Introduction

E-learning is a modern, developing way how to approach to education using opportunities of computers and the Internet. These conveniences 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. Learners may be from different parts of the world, of different age, education or skills. The ease and relative effectiveness of e-learning systems caused their big development in the last two decades – computer educating tools exist in almost all fields of school education. 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, variety of possible learning mechanisms, and novelty of the e-learning as a subject of science, the processes how to obtain this needed knowledge are not standardized.

The aim of this thesis is 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 RoboMission e-learning system as a base for our research; all analyses are performed on RoboMission data. Results of this research are summarized into a form of 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.

In order to properly specify the information which we are interested in, we defined seven research questions which cover various aspects of our problematics:

1. What is the difficulty of a task?

2. What is the complexity of a task?

3. How unique are the correct solutions of a task?

4. Is a task similar to some other task?

5. What problems do learners have while solving a task?
6. What is the learner's performance on a task?
7. What is the learner's performance in the whole system?

The motivation for this research is a better understanding of the properties of RoboMission's tasks and learners and thus a solid base for improvement of the whole system. The analytic process developed in this thesis is transferable to other introductory programming e-learning systems where it can be applied after changes applied with respect to a systems' specific features.

The structure of this thesis is as follows: In Chapter 2 we describe the problematics of the introductory programming e-learning, measures used for a description of phenomena in systems of this type, and the visualization of data in the form of dashboards. Chapter 3 concerns with our source system RoboMission, description of RoboMission data we process, and the procedure of processing the data. The tasks' characteristics are discussed in Chapter 4, the learners' characteristics in Chapter 5.

2 Related Work

Since the e-learning system creation requires people who are competent in the field of computer science, programming is one of the core domains of the e-learning systems' interest.

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

The second dividing criterion is the environment in which a program output is processed. Classical console environment which is used in 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. Frequent graphical output environments are two. Drawing interface, in which a learner controls a pen (often represented by a turtle and its tail) by

Table 2.1: Examples of Introductory Programming E-learning Systems developed at Faculty of Informatics, Masaryk University

system	language	code construction	output interface	loops	conditions	variables	functions	recursion
RoboMission	Python-like	block	grid	*	*			
Turtle Graphics	textual	block	drawing	*		*		
Robotanist	pictorial	block	grid				*	*
Interactive Python	Python	textual	console	*	*	*	*	?

series of distance and direction-change commands, and grid interface, where a robot walks and performs actions on squares of a grid game board.

This programming e-learning systems division can be demonstrated on the systems developed by Adaptive Learning research lab at Faculty of Informatics, Masaryk University, Brno, Czechia[1], as seen in table 2.1.

2.2 Measures

In order to obtain information about measures and characteristics used for displaying of a description of tasks and learners data in the field of introductory programming e-learning systems, a 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 2.2, 2.3 and 2.4. This set comes from e-learning systems of various types of content and logging and therefore only a part of these items is

1. Website http://https://www.fi.muni.cz/adaptivelearning/ .

at least theoretically usable in the context of RoboMission – the usable ones are marked as "relevant".

2.3 Dashboards

During a research of so far created introductory programming dashboards in literature, there was found no dashboard of our type, i. e. designed for non-real-time, once-in-a-while use.

REAL-TIME OR OTHER DASHBOARDS: DIANA, MURPHY, MATSUZAWA-OSA, MATSUZAWA-PROGRAMMING

DASHBOARDS IN GENERAL: PODGORELEC, SANTOS, FEW

CA. 1 PAGE

Table 2.2: Found measures and characteristics, part 1/3

no.	name	papers	relevant
TIME			
01	working time	[1], [2], [3], [4]	*
02	time between submits/compilations	[5]	*
03	time between compilation errors	[5]	
04	percentage of time in non-compiling state	[1], [2]	
05	compile error correction time	[2]	
06	block-editing-mode time ratio	[2]	
KEYSTROKES, CHARACTERS, LINES			
11	total keystrokes	[1]	
12	percentage of keystrokes in non-compiling state	[1]	
13	total lines of code	[1], [2], [6], [7]	*
14	code update pattern[a]	[8]	
15	time change of code update pattern	[8]	
OPERATIONS			
21	particular submits/compilations	[3]	*
22	total submits/compilations	[3], [9]	*
23	total successful/unsuccessful compilations	[3]	
24	percentage of successful/unsuccessful compilations	[3]	

a. Number of lines added/deleted/modified.

Table 2.3: Found measures and characteristics, part 2/3

no.	name	papers	relevant
CODE CONTENT			
31	total control flow statements	[1], [6]	*
32	total function definitions	[6]	
33	total variables	[6]	
ERRORS			
41	particular compilation errors	[3]	
42	most common compilation/run-time errors	[3]	
43	total compilation errors	[3]	
44a	error quotient[a]	[5], [10], [11]	
44b	Watwin (error quotient improvement)[b]	[10], [11]	
45	error category	[4]	*
SCORE			
51	score/points/grade	[5], [6], [8], [10], [12], [13], [14]	*
CORRECTNESS			
61	correctness	[4], [10]	*
62	tests passed	[10]	
UNIQUENESS			
71	unique unit test outputs	[9]	
72	unique AST	[9]	*

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

Table 2.4: Found measures and characteristics, part 3/3

no.	name	papers	relevant
TASK SIMILARITY			
81	bag of words of task description	[8], [13]	*
82	API calls similarity	[8], [13]	*
83	abstract syntax trees similarity	[8], [13]	*
84	concepts contained similarity	[15]	*
85	Levenshtein distance	[15]	*
86	n-grams overlap	[15]	*
87	input-output behavior similarity	[15]	*
88	program dependence graph structure	[15]	
89	code-state transition graph node similarity	[12]	*
MODE OF STATISTICS			
91	absolute stats for learner	all	*
92	relative stats for learner[a]	[3]	*
93	absolute stats for class	[3]	

3 Data and data processing

SOMETHING

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 [16]. It learns 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 3.1.

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) and a block-based length-of-program limit.

Tasks are grouped into 9 levels, each level consists of 3 phases based on the task difficulty. Learners practice new programming concepts or their combination in each level. Adaptivity is provided by a recommendation of the next task – a learner is guided through the system's levels and phases. A learner's performance on each answered task is evaluated by one four labels based on correctness and solving time – incorrect, poor, good and excellent. Each of the three latter 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 [16].

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

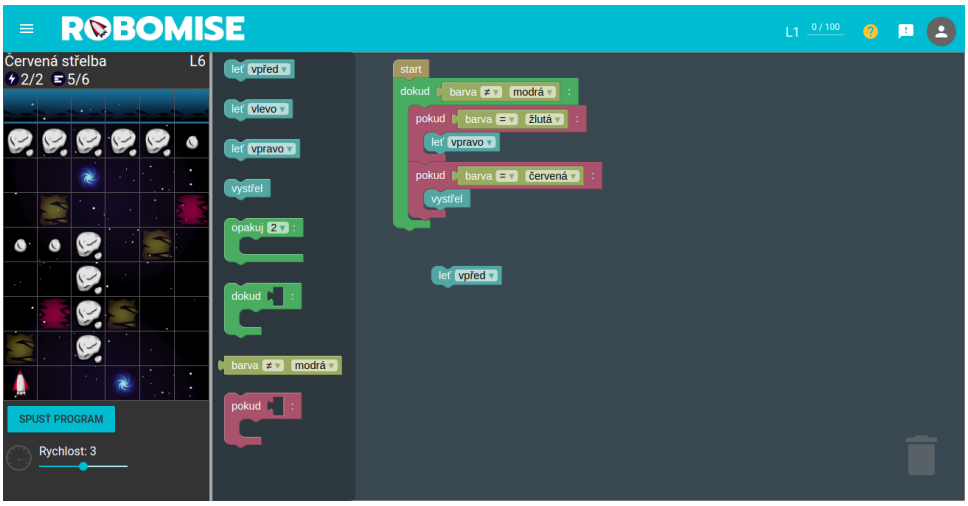


Figure 3.1: RoboMission system snapshot

Table 3.1: Data statistics

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

3.2 Data

Our data consist of three .csv files which describe tasks, learner's 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 3.2 and 3.3.

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 execution of the code; if the code is correct, the task is considered solved and the next task is recommended to the learner.

REST OF DATA

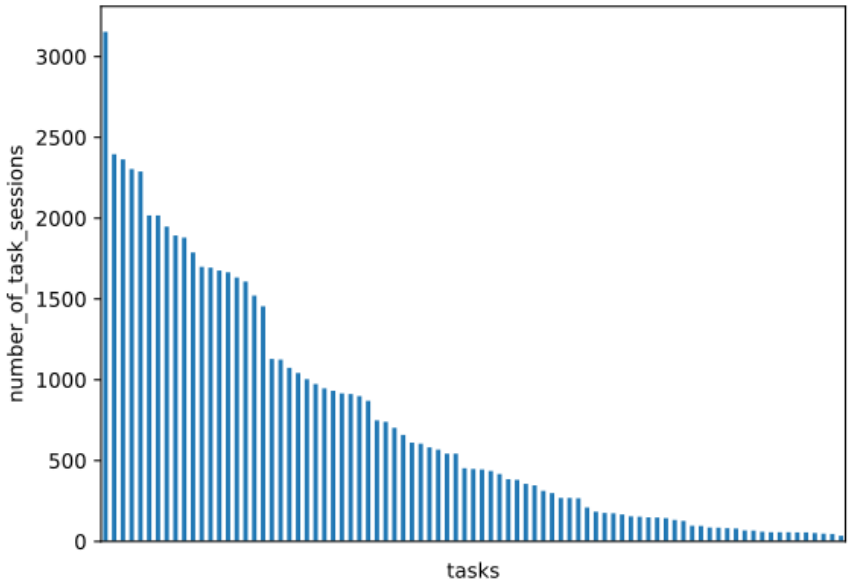


Figure 3.2: Number of task sessions for each task

The first thing needed for proper processing of data was a slight modification of all logged programs – the shortened version of the program code (called *MiniRoboCode*) which is used in log files, uses a one-character abbreviation for each type of blocks. E. g. `"f"` for `"forward"` or `"Ix>3{l}"` for `"if position > 3: {left}"`. The problem is that symbol `"r"` is used for two blocks – `"right"` and `"red"` (test for the red color of the background). This ambiguity causes problems in MiniRoboCode 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 it is very time-consuming. Our recommendation for RoboMission developers is to remove this ambiguity in logged files and to change settings of MiniRoboCode creating software for future logs.

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

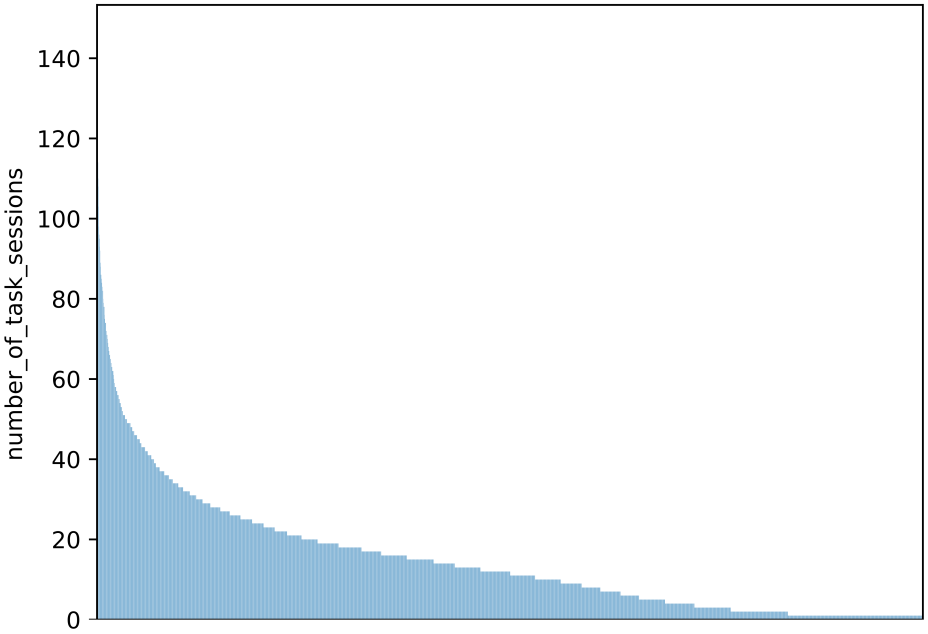


Figure 3.3: Number of task sessions for each learner

During an investigation, two causes of this discrepancies were found. The first one is 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 refer to the new task now. This caused a 98.5 % of rejected correct programs. The second cause is more complicated. The code submitted by a learner is sent to the interpreter to be evaluated. However, a graphical web interface needs an code evaluation block-by-block in order to highlight currently processed blocks of the learner's code. This asynchrony may result in problems if a non-standard event happens in the communication between the learner's device and 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 no proof of this was found.

As a result of found discrepancies, a synchronous Python interpreter was written for MiniRoboCode. A new information about the correctness of a program and trajectory of the spaceship through the game board was obtained; when the correctness of a program code is mentioned, the results from the synchronous interpreter are meant. This new interpreter was provided to the author of RoboMission for purposes of better RoboMission functionality.

3.3 Methodology

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

RoboMission specifics. Particular measures are described in sections corresponding to each research question.

All the measures were then computed on RoboMission data and some of them were 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.

There are two main correlation coefficients – the Pearson's and the Spearman's ones. Both of them were computed and compared. In many cases, they perform in a similar way 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 showed significantly higher correlation than the Pearson's one. In order to objectively quantify this difference, a correlation of measure correlation tables was computed. Here, only one correlation coefficient, the Pearson's one, was chosen for simplicity. However, there is one more dimension to decide how to compute the correlation of correlation tables – 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 this computation were performed. The first one was computing with full correlation tables, the second one only with elements under the main diagonal. In the end, four kinds of correlations were computed for each research question's measures set – Pearson's correlation of measures, Spearman's correlation of measures, Pearson's correlation of full correlation tables and Pearson's correlation of lower triangle correlation tables. All of these correlation tables are presented in APPENDIX KOLIK?.

The analyses showed that the difference between the correlation of the full and lower triangle correlation matrices may be significant, the full matrix method was always producing higher values; therefore

the lower triangle matrix method was used. This method showed that the choice of the correlation coefficient plays a role at three research question sets (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 determine whether two measures are similar or not.

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

4 Task Characteristics

SOMETHING

4.1 Task Difficulty and Complexity

Our concept of a difference between task difficulty and task complexity follows on from the work of Sheard et al. [7]. While task difficulty lies in total demands necessary to complete a task, task complexity concerns only with cognitive demands. The difficulty is thus always greater than complexity. Non-cognitive aspects which determine the relationship between difficulty and complexity are the length of task, task description, and solving environment.

Information about difficulty and complexity of tasks may have an impact on the categorization of tasks into levels and phases, on the creation of new tasks and modification of current tasks with extreme difficulty or complexity measures.

We selected nine measures which are associated with task difficulty 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.

Task complexity measures which are independent of the other aspects of difficulty were selected as follows:

- ratio of unsuccessful task sessions,
- number of distinct blocks used in task sample solution,
  - treat all block types as distinct (abbreviated as *blocks 1*),
  - treat *forward*, *left* and *right* blocks as one type (*blocks 3*),
  - treat *forward*, *left*, *right* and *shoot* blocks as one type (*blocks 4*).

The *ratio of unsuccessful task sessions* is computed as the number of task sessions labeled as unsuccessful, divided by the number of all logged task sessions.

The *number of distinct blocks used in task sample solution* is a proxy measure for the number of concepts which are necessary for successful

completion of a task. Various variants of this measure were created because blocks *forward*, *left* and *right* belong all to the single basic concept "moving" and semantics of block *shoot* is very similar to the previous three ones (compared to the others, control statement blocks).

Selected task difficulty measures which are influenced by non-cognitive aspects of difficulty are these ones:

- median of solving times,
- median of number of edits,
- median of number of submits,
- median of length of correct solution,
- length of sample solution,
- median of deletions,
  - number of all blocks deleted (*deletions all*),
  - number of all deleting edit operations (*deletions edits*),
  - deletion used (binary variable, *deletions 1/0*),
- ratio of learners who were deleting during the task solving.

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

Medians of number of edits and submits are medians of number of operations the learners performed, filtered by type of operation – modification of program code and its execution.

Median of length of correct solution and *length of 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.

Deletions measures are based on the number of shortenings of the program code. There are several ways how to compute deletions – we selected three of them. The most simple 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 - 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 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 given task.

In case of edits and deletions, we should remark that although a learner wants to modify a program by a single action (change of one block), logs of this action may lead to a number of operations different from 1. Both reasons for this state come from the properties of RoboMission 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 the block with a single operation because every time a learner manipulates with a block, he/she manipulates also 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 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 case of counting all blocks or 2 in case of counting deleting edit operations or 1 in case of binary use of deletion.

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 in 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 and n, 1 and 1 if the given block is removed from the second segment.

Results of correlation analysis can be seen in figure 4.1 – it is a Spearman's correlation table dendrogram. Both complexity measures (and their variants) do not correlate with each other significantly better

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

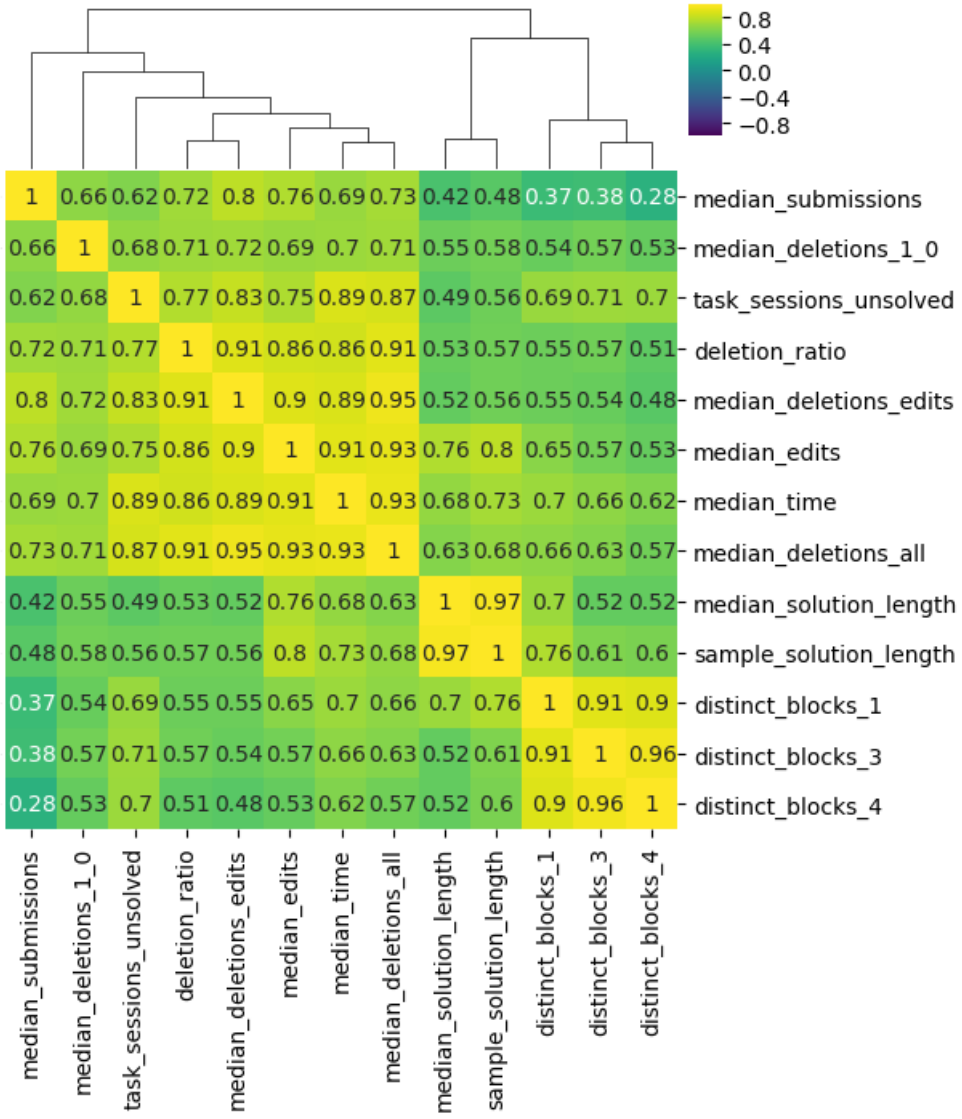


Figure 4.1: Difficulty and complexity measures – Spearman's correlation

Table 4.1: Difficulty and complexity measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
ratio of successful task sessions	0.035	0.204	0.584	84
number of distinct blocks 1	3	4	7	8
number of distinct blocks 3	1	3	6	6
number of distinct blocks 4	1	3	5	6
median of solving times	27	119	292	79
median of number of edits	4.0	13.5	22.6	35
median of number of submits	1	2	4	9
median of length of correct solution	3	6	10	11
length of sample solution	3	6	11	15
median of deletions all	0	5	15	25
median of deletions edits	0	2	5	11
median of deletions 1/0	0	1	1	2

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 treat them all further as measures of single tasks aspect and research question No. 2 consider as answered together with question No. 1.

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

Distinct blocks measures are strongly dependent on the level to which a task belongs. Since the number of practiced block types grows with the number of level, the information contained in distinct block count can be successfully estimated from the task's level information

which does need to be computed. Therefore this group is not used for the purposes of the dashboard, the level information is used instead.

Median correct solution length correlates with 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. Sample solution length can be easily found out from the task's sample solution and that is why this group is not processed further.

The rest of measures is not derivable from static task characteristics. The deletion measures and median of submits range only 6 or fewer values for the lower half of tasks (table 4.1). The number of edits is distorted by the problems mentioned above and the ratio of unsuccessful task sessions suffers from 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 measure is easily understandable, straightforwardly computable and its range is sufficiently wide and thanks to the application of median, it copes with possible time measuring problems. Therefore this measure is selected for use on the dashboard.

4.2 Solution Uniqueness

Correct solutions measures express how diverse correct solutions of 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 analysis whether learners use concepts which are intended to practice. If a task has many dissimilar correct solutions, it may be recommended to think about reworking the task.

Solution uniqueness measures are these:

- number of unique correct solutions,
- entropy of unique correct solutions distribution,

- number of unique visited squares sequences,
- entropy of unique visited squares sequences distribution,
- number of correct solutions clusters.

The *number of unique correct solutions* expresses 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. output behavior of program `"while color != blue: {something}"` is at many tasks identical as of `"while color == black: {something}"` since 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. While evaluating programs, sequences of squares visited by the spaceship are recorded. 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 n-ary *entropy* [17]:

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

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 distribution of correct solutions as the entropy of unique correct solutions distribution and the entropy of unique visited squares sequences distribution.

The *number of clusters* measure uses 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 by many ways, e. g. by a `for` loop or by a `while` loop which

Table 4.2: Solution uniqueness measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
number of unique correct solutions	2	8	32	27
entropy of unique correct solutions distribution	0.174	0.642	0.891	81
number of unique visited squares sequences	1	1	5	11
entropy of unique visited squares sequences distribution	0.000	0.000	0.717	42
number of correct solutions clusters	1	1	9	12

may differ in various stop conditions of the same meaning. Usage of different code elements often implies different idea behind the code so the distinguishing these ideas may be valuable.

Our clustering is based on abstract syntax trees (ASTs) and tree edit distance (TED). ASTs are hierarchically clustered into groups with stopping parameter of cophenetic distance. 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 constructed iteratively, ordered by 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. 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. Distance value 5 was also used by Huang et al. in [9]. The number of clusters which were created by this procedure is taken as a measure. Created AST builder program was provided to the author of RoboMission for free use.

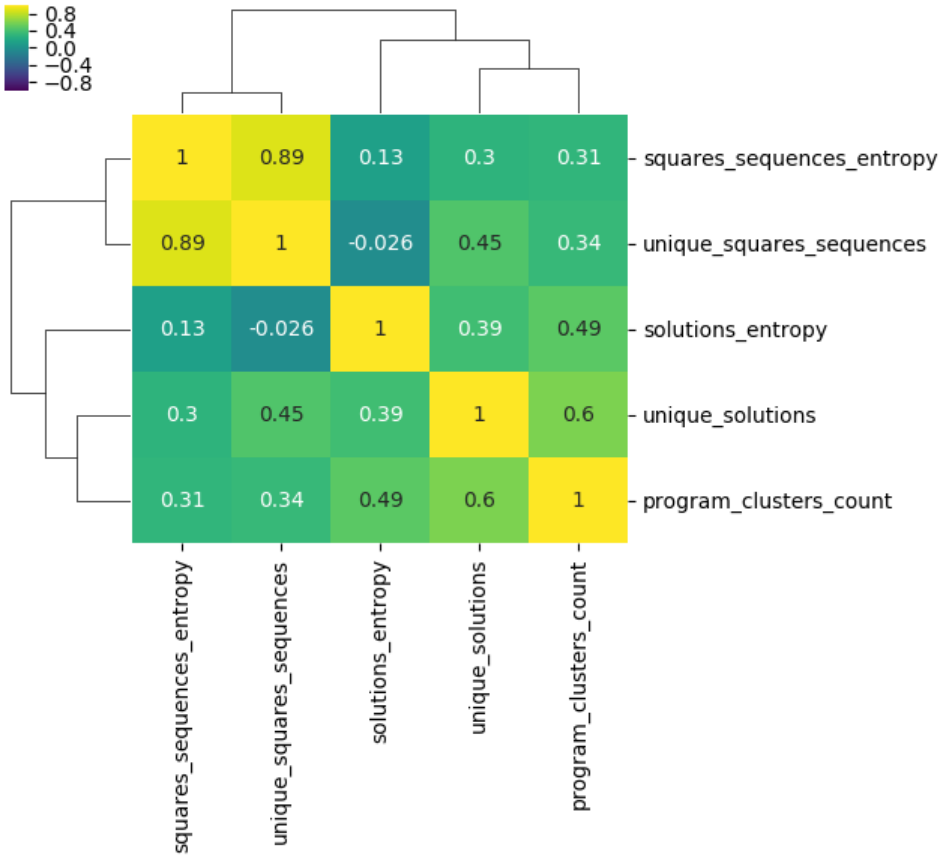


Figure 4.2: Solution uniqueness measures – Spearman's correlation

Results of correlation analysis are pictured in figure 4.2.

Measures of unique visited squares sequences (its number and 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 (table 4.2). Therefore the entropy is zero and thus correlation is high. The other measures do not form strongly correlated groups.

Both entropy measures are not easily understandable. In case of the number of unique visited squares sequence, the entropy of unique visited squares sequence distribution and the number of correct solutions clusters measures, more than half of tasks is described by the same value (1, 0 and 1). Therefore we select the number of correct solutions to the final dashboard. However, since the correlation of this measure is not strong, some information represented by other measures would be lost. In order to take advantage of this data and to strengthen their information value, another feature is added to the dashboard – complete view of all correct visited squares sequences seen so far accompanied with their frequencies, represented by the most frequent solution which leads to the given visited squares sequence for clarity.

4.3 Task Similarity

Levels and phases in RoboMission are practiced 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 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 legit solitary task which could be accompanied by new tasks similar to this one.

We selected several source task similarity variables, however, it is much easier to measure distance (or dissimilarity) of task features. Therefore all the selected variables work in a reverse way – the higher the value, the more dissimilar tasks. Selected source task dissimilarity variables are these ones:

- sample solutions similarity,
  - abstract syntax trees – tree edit distance,

  - bag-of-blocks – Euclidean distance,
  - shortened code – Levenshtein distance,

- game world similarity,
  - bag-of-game-world-entities – Euclidean distance.

The selected variables are used for computation of task dissimilarity matrix with size $n \times n$ 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:

- distance to the closest task,
- how many tasks are situated within $2^{nd}/5^{th}/10^{th}$ percentile of distances.

Sample solutions similarity variables are based on sample programs – their features are processed in various ways and then compared. In the first variable, a task's sample solution is transformed into its abstract syntax tree and tree edit distances of task pairs are computed. The second variable is based on counting occurrences of blocks – vector of these occurrences is called bag-of-blocks, the Euclidean distance of this vectors is used as a feature. Levenshtein distance is applied to plain programs in order to obtain the third variable – addition, removal or change of a character of the first program in order to modify it to the other program is counted as +1 to the distance of tasks. The *game world similarity* variable describes the number of occurrences of all entities present on the game board – asteroids, diamonds, wormholes, colorful squares etc. – in the form of vector. The dissimilarity of these vectors is then computed by the Euclidean distance. The possibility of covering even the position of game world entities would be methodologically very complicated, therefore this information is not used.

Distance to the closest task concerns only with the closest task according to a given variable, information about all other distances of given task is discarded. The latter way computes 2^{nd}, 5^{th} and 10^{th} percentile of all distances in the 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 a 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 count tasks similar to the given task.

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

The game world entities measure concerns with the specification of a task. Thus, it is totally independent of the task solution. Moreover, this measure works only with occurrences of the entities, not with their position – as every chess-player knows, the position of chess pieces does play a role.

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 which are very dissimilar to each other – one task would be a source one, the second would contain many redundant entities in the areas which are unreachable or not supposed to be visited. If we forced maximal simplicity of the game world, a game boards similarity could be related with a difference of task difficulties but there is no such restriction. Sometimes, the presence of technically redundant entities is a developer's intention to aesthetically improve the task. For this reason, we do not classify game world entities measures as informative.

The bag-of-blocks measures do not take account of a structure too, it works only with vectors of block 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 together, we deduce that other measures which reflect the structure of the code would be more informative.

Those measures form the third correlated group. AST TED works with full information about a program code structure while the textual form of the program (used by Levenshtein code) flattens the syntax tree into the sequence of characters but we can see that corresponding pairs of Levenshtein code and AST TED measures correlate very strongly with each other and even the basic statistics (table 4.3) of these two measures groups are very similar. This brings us to the finding that there is very little difference between Levenshtein code and AST TED

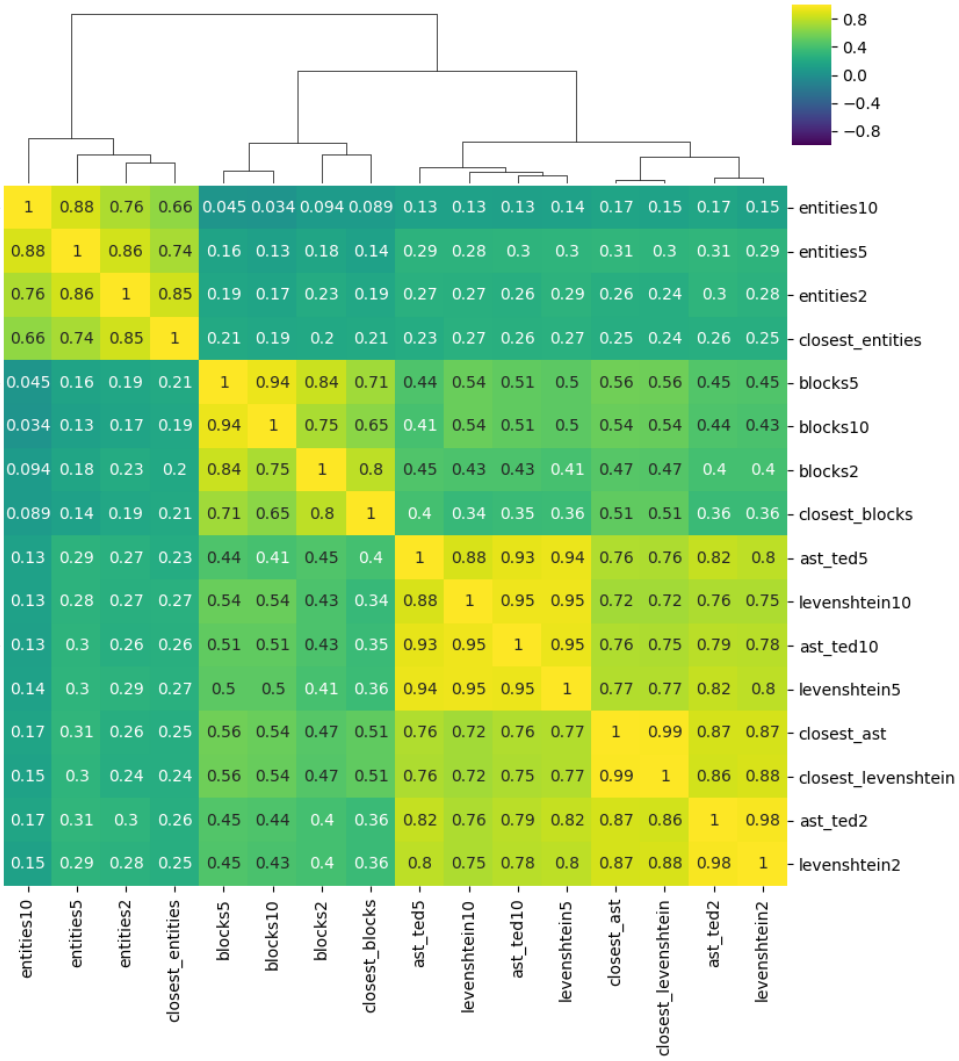


Figure 4.3: Task similarity measures – Spearman's correlation

Table 4.3: Task similarity measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
AST TED, 2^{nd} percentile	0	1	9	9
AST TED, 5^{th} percentile	0	2	15	17
AST TED, 10^{th} percentile	0	8	22	23
Euclidean bag-of-blocks, 2^{nd} percentile	0	2	7	0
Euclidean bag-of-blocks, 5^{th} percentile	0	6	12	17
Euclidean bag-of-blocks, 10^{th} percentile	1	11	18	22
Levenshtein code, 2^{nd} percentile	0	1	9	9
Levenshtein code, 5^{th} percentile	0	3	18	18
Levenshtein code, 10^{th} percentile	0	6	22	20
Euclidean bag-of-entities, 2^{nd} percentile	0	1	5	8
Euclidean bag-of-entities, 5^{th} percentile	0	4	9	13
Euclidean bag-of-entities, 10^{th} percentile	0	9	17	21
AST TED, closest task	1	3	9	15
Euclidean bag-of-blocks, closest task	0.0	1.4	2.0	9
Levenshtein code, closest task	1	3	9	15
Euclidean entities, closest task	2.8	5.5	12.1	48

measures groups and thus we can eliminate AST TED measures as much more difficult to compute.

From the remaining measures, we feel that 9 values for all the tasks and 3 values for the half of them is too few for a proper understanding of the similarity of a task to all the others. We selected Levenshtein code, 5^{th} percentile for the dashboard; the 10^{th} percentile does not bring a significant change of the measure's range and lower values of variables are easier to handle for a human.

The previous three measures expressed how much a task is similar to all the others, the remaining measure tells us how much a task is similar to the most similar task. This is a different aspect of similarity and therefore we include the dissimilarity of the closed task by Levenshtein code into the dashboard.

4.4 Frequent Problems

The key point of the research question No. 5 is the definition of frequent problem state term.

Recognition of a state in which a learner is in trouble is not easy. Perception of troubles is subjective, a trouble state of one learner may be a totally problem-free state for a different learner who has different learning style. Recognition of learning styles is difficult too because the majority of RoboMission learners tries to solve too little number of tasks. For these reasons, the problem state has to be defined universally. The selected definition understands problems as a subset of incorrect submits or unsuccessful task sessions.

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

Selected frequent problems characteristics are these ones:

- list of frequent incorrect submits,
- list of frequent leaving points.

These two characteristics are not measures. We came out of the assumption that merit of an answer to the research question No. 5 is in the identification of the problem points, not in quantifying them.

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

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

The process of setting thresholds for frequent problems was considering both absolute and relative frequency if items. The main criterion of frequentness is the relative ratio of a problem's occurrences to all occurrences of task's problems. However, the size of data for each task differs a lot - the least-solved tasks have only around 70 wrong submits and around 10 leaving points while the most-solved ones have over 10,000 wrong submits and over 300 leaving points. In the case of tasks with the smallest amounts of data, only a relative threshold may lead to "frequent" problems with only small units of occurrences. Therefore a basic absolute condition to frequent problems is needed.

Data analysis showed that in 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[2] was below 10 % for every value of the absolute threshold. The same result was in 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.

2. 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

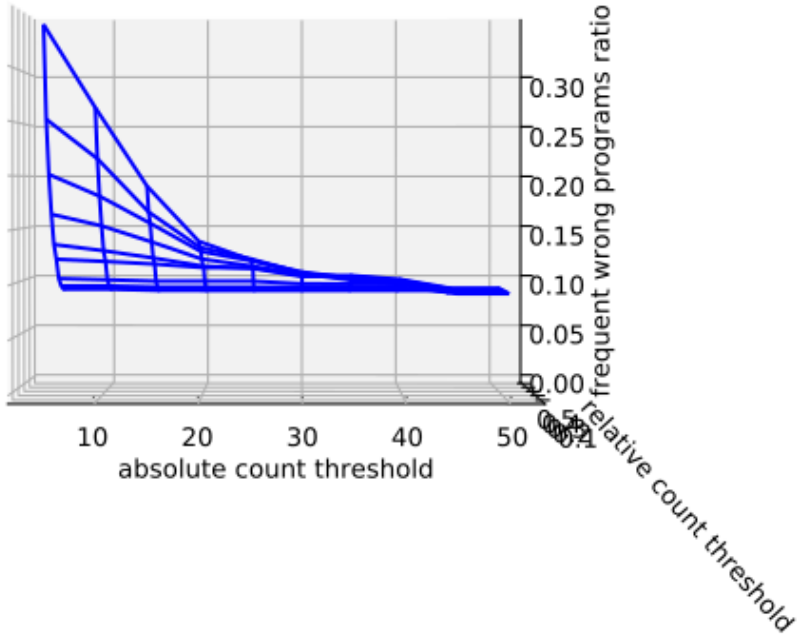
For this reason, some kind of programs grouping is needed. We selected grouping based on output behavior equivalence, i. e. on the equivalence of visited squares sequences. The results of grid testing of absolute and relative thresholds on grouped programs can be seen in figures 4.4 and 4.5. The vertical lines depict the selected frequency type values while the horizontal or decreasing lines depict the other one. The total relative frequency of frequent problems in the whole dataset is represented by the values of intersections on the vertical axis.

Based on this analysis, we set thresholds for frequent problems groups as in table 4.4. These thresholds imply the total relative frequency of frequent problems equal to 10 % at incorrect submits, respectively 23 % at leaving points. This values of thresholds were selected in order to satisfy the requirements that frequent problems should be frequent in both tasks and total scale. However, there are even more values which satisfy the given requirements and these ones were selected based on our subjective judgment.

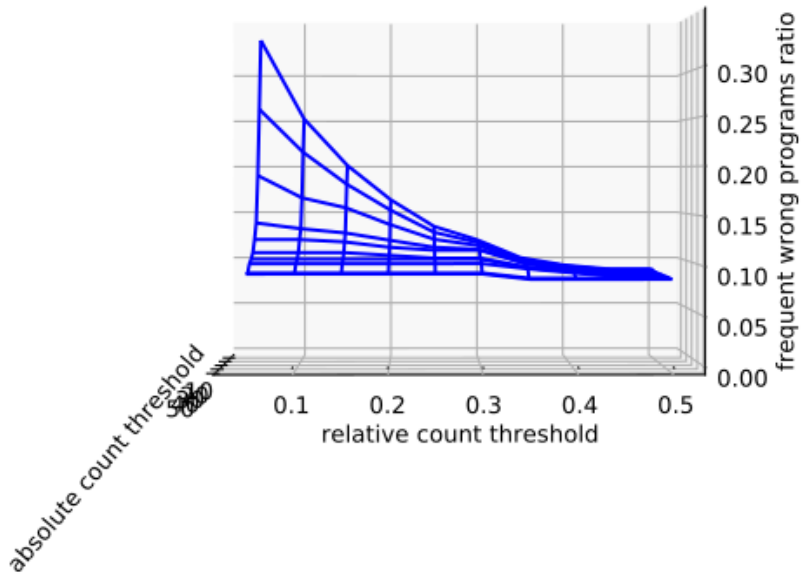
We created 4 measures in order to explore relations between our two characteristics with thresholds set as above:

- number of unique frequent incorrect submits,
- number of unique frequent leaving points,
- ratio of frequent incorrect submits,
- ratio of frequent leaving points.

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

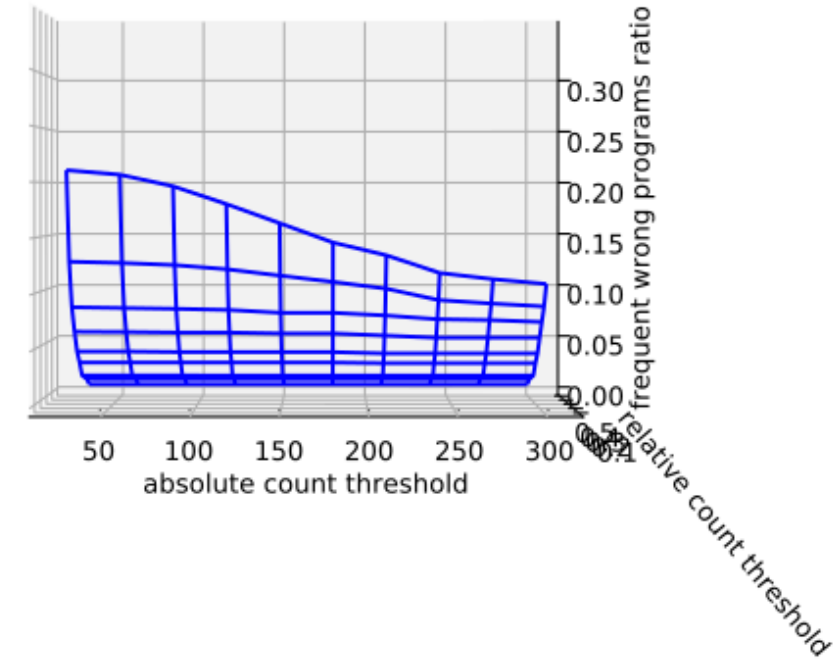


[a]

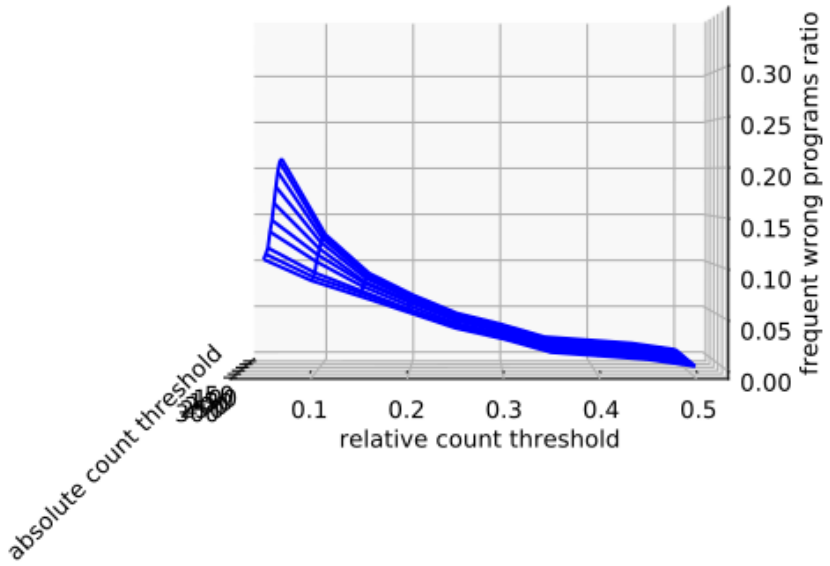


[b]

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

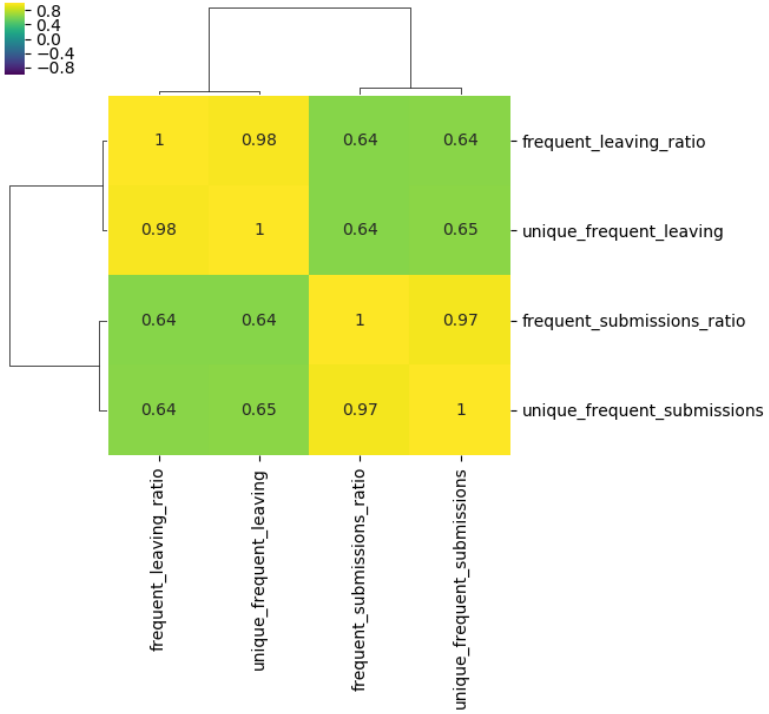


Figure 4.6: Frequent problems measures – Spearman's correlation

Table 4.5: Frequent problems measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
number of unique frequent incorrect submits	0	1	3	5
number of unique frequent leaving points	0	0	2	4
ratio of frequent incorrect submits	0.000	0.000	0.603	50
ratio of frequent leaving points	0.000	0.000	0.277	33

The results of analyses are presented in figure 4.6 and table 4.5.

It is clearly visible from figure 4.6 that both characteristics (and their accompanying measures) are correlated with each other only slightly and therefore both of these characteristics could be informative in its own way. On the other hand, table 4.5 shows that even with these very soft conditions 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 points are listed in the dashboard, grouped by their visited squares sequences and represented by the most frequent program belonging to the given visited squares sequence.

5 Learner Characteristics

ONE OR TWO PARAGRAPHS OF TEXT

5.1 Task Session Performance

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 evaluation or a learner's performance on a task session is the correctness of 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/her to many dead ends. The similar principle holds in case of unsuccessful learners who got stuck only a step by the correct solution and those who give up in the beginning.

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

- solving time,
- number of edits,
- number of submits,
- number of deletions,
  - number of all blocks deleted (*deletions all*),
  - number of all deleting edit operations (*deletions edits*),
  - deletion used (binary variable, *deletions 1/0*).

Solving time measure is a time in seconds for which a learner was solving a task session. This measure may be distorted by possible

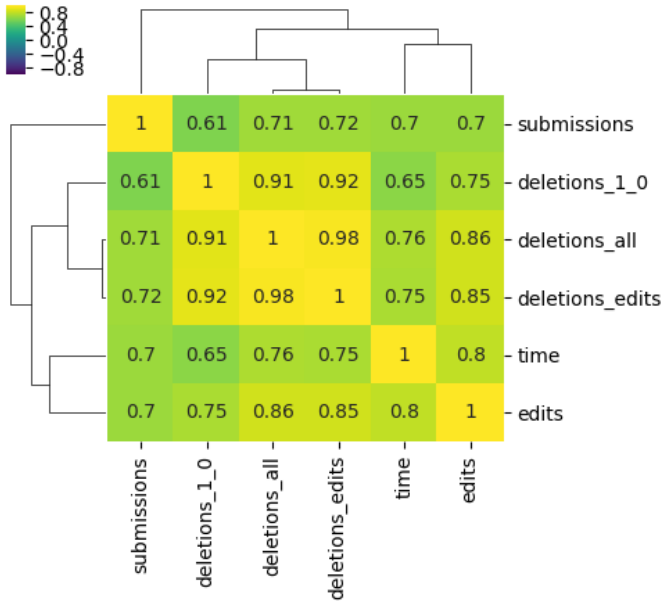


Figure 5.1: Learner's task session performance measures – Spearman's correlation

overvaluation if a learner was doing some other things while solving a task (chatting with friends, being away from keyboard etc.).

Number of edits and *number of submits* measures express the numbers of code-change and execution operations. We remind a remark from Section 4.1 that not every single modification action results in one edit operation. It may be less if a learner modifies blocks inside the code and it may be more 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 in with `start` block at the very end – this would lead to solving the entire task by a single logged edit operation.

Number of deletions measures are computed from edit operations in the same way as in Section 4.1.

The results of correlation analysis are shown in figure 5.1. We can see that all measures correlate with each other at least at 0.6.

Deletion measures correlate strongly with each other. The range of binary deletion measure is too narrow (table 5.1), the other two

Table 5.1: Task session performance measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
solving time	15	60	336	2242
number of edits	3	8	29	220
number of submits	1	2	8	90
number of deletions all	0	1	18	217
number of deletions edits	0	1	7	97
number of deletions 1/0	0	1	1	2

deletion measures depend on the number of edits – if we know the length of correct solution and number of edits, we can quite precisely estimate number of deletion edits and then even the number of all deleted blocks since they are strongly correlated, so the number of edits would be a better choice than these measures.

From the remaining three measures, the number of edits may be distorted by problems with logging, solving time by focusing on different activities. However, this distortions are not so often because otherwise, the measures would not correlate. The range of 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 solving time measure to the final dashboard.

5.2 Overall Performance

The research question No. 7 concerns with learners' performance in the whole system. This information is valuable for the learner model used in the system – information about 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

learner's task sessions performances. However, a single-number metric for a quick overview is lacking.

We selected these measures for the analysis:

- total number of solved tasks,
- total number of points gained,
- total number of unique blocks used in correct solutions,
- total solving time.

The *total number of solved tasks* measure is a straightforward way how to measure learner's progress – if we suppose that all learners walk through RoboMission based on system's recommendations, the more difficult tasks a learner solves, the higher total number of solved tasks is. On the other hand, the current recommendation system forces worse learners to practice 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 computed as the sum of tasks solved in each level multiplied by the number of the corresponding 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 its knowledge.

Total solving time is a sum of solving times of all learner's task sessions. This measure ma be distorted in similar was as in Section 5.1 but this time the influence of a single distortion is reduced by summing all learner's solving times.

Results of correlation analysis are displayed in figure 5.2, all measures correlate with each other at more than 0.7 so the other criteria select the one for the dashboard. The used blocks measure ranges only 10 values (table 5.2), the total solving time and the total number of solved tasks does not take the difficulty of tasks into consideration. Therefore we selected the total number of points gained for the dashboard.

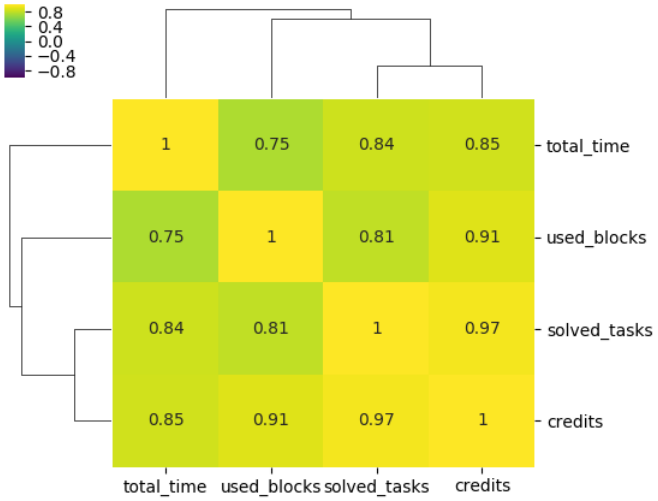


Figure 5.2: Learner's overall performance measures – Spearman's correlation

Table 5.2: Overall performance measures statistics

measure	0.1 quantile	median	0.9 quantile	unique values
total number of solved tasks	2	12	29	72
total number of points gained	2	21	86	220
total number of used blocks	3	5	8	10
total solving time	87	1199	4687	2265

6 Dashboard

DESCRIPTION OF DASHBOARD, CA. 2 PAGES
USABILITY FOR OTHER SYSTEMS (Discussion?)

7 Conclusion

WE PERFORMED ANALYSES AND MADE A DASHBOARD, 1-2 PAGES

A Appendix

CODE
    DASHBOARD

Bibliography

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.

8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.

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

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